Geoinformatics | Course Remote Sensing (1)

Schmitt | Ulloa

Summer Semester 2020

Practice 6: Accuracy Assessment

Overview

Objectives: Create your validation data and perform an accuracy assessment of the classification results from the last practice.

Data: For this practice, use the following files:

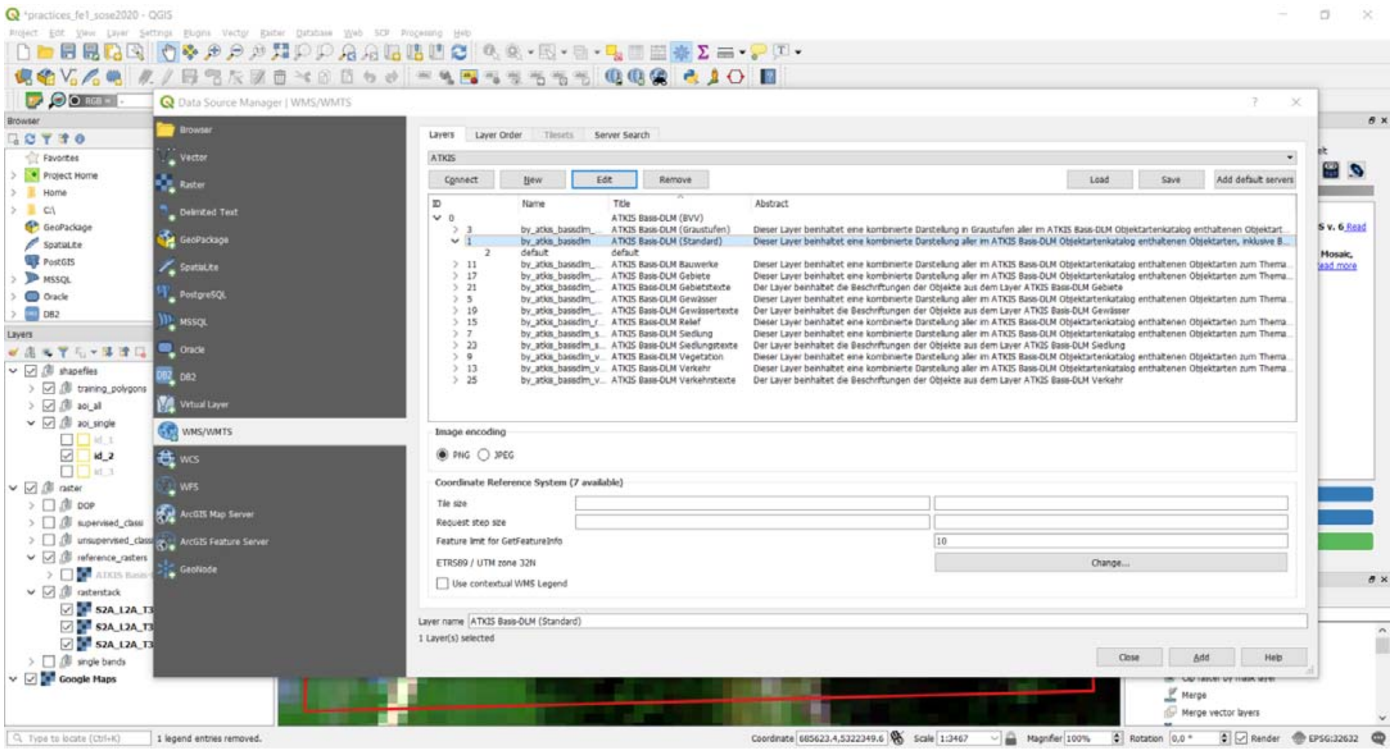
- Raster files:
  - Image classified: GeoTIFF Sentinel-2A Rasterstack (S2A_L2A_T32UPU_rstck_id1.tif, S2A_L2A_T32UPU_rstck_id2.tif or S2A_L2A_T32UPU_rstck_id3.tif, depending of the AOI you were assigned.)
  - Classification: GeoTIFF raster with the classification result using either RF or MD.
  - Reference data: ATKIS and CORINE datasets.
- Vector file: your validation polygons in Shapefile format.

Tasks: load your S2A rasterstack and reference data in QGIS to create your validation polygons. Use your validation polygons to run an accuracy assessment on your classification rasters. Compare results and understand the different types of errors used to quantify the quality of your classification.

Procedure

A. In QGIS: create validation data

1. Load your Sentinel-2A Rasterstack in QGIS. On top of that, load ATKIS and CORINE datasets. To open ATKIS raster, add it as a WMS Layer. Enter username and password and connect to the server. Check that the projection is UTM 32N. Load the 'Standard' band to visualize all features. If you want, you can also load individual features, like water bodies ('Gewässer') or vegetation, for example.



ID	Name	Title	Abstract
0		ATKIS Basis-DLM (BVV)	
3	by_atkis_basisdlm_...	ATKIS Basis-DLM (Graustufen)	Dieser Layer beinhaltet eine kombinierte Darstellung in Graustufen aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektart...
1	by_atkis_basisdlm	ATKIS Basis-DLM (Standard)	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten, inklusive B...
2	default	default	
11	by_atkis_basisdlm_...	ATKIS Basis-DLM Bauwerke	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
17	by_atkis_basisdlm_...	ATKIS Basis-DLM Gebiete	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
21	by_atkis_basisdlm_...	ATKIS Basis-DLM Gebietstexte	Der Layer beinhaltet die Beschriftungen der Objekte aus dem Layer ATKIS Basis-DLM Gebiete
5	by_atkis_basisdlm_...	ATKIS Basis-DLM Gewässer	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
19	by_atkis_basisdlm_...	ATKIS Basis-DLM Gewässertexte	Der Layer beinhaltet die Beschriftungen der Objekte aus dem Layer ATKIS Basis-DLM Gewässer
15	by_atkis_basisdlm_r...	ATKIS Basis-DLM Relief	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
7	by_atkis_basisdlm_s...	ATKIS Basis-DLM Siedlung	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
23	by_atkis_basisdlm_s...	ATKIS Basis-DLM Siedlungstexte	Der Layer beinhaltet die Beschriftungen der Objekte aus dem Layer ATKIS Basis-DLM Siedlung
9	by_atkis_basisdlm_v...	ATKIS Basis-DLM Vegetation	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
13	by_atkis_basisdlm_v...	ATKIS Basis-DLM Verkehr	Dieser Layer beinhaltet eine kombinierte Darstellung aller im ATKIS Basis-DLM Objektartenkatalog enthaltenen Objektarten zum Thema...
25	by_atkis_basisdlm_v...	ATKIS Basis-DLM Verkehrstexte	Der Layer beinhaltet die Beschriftungen der Objekte aus dem Layer ATKIS Basis-DLM Verkehr

Image encoding: PNG [✓] JPEG

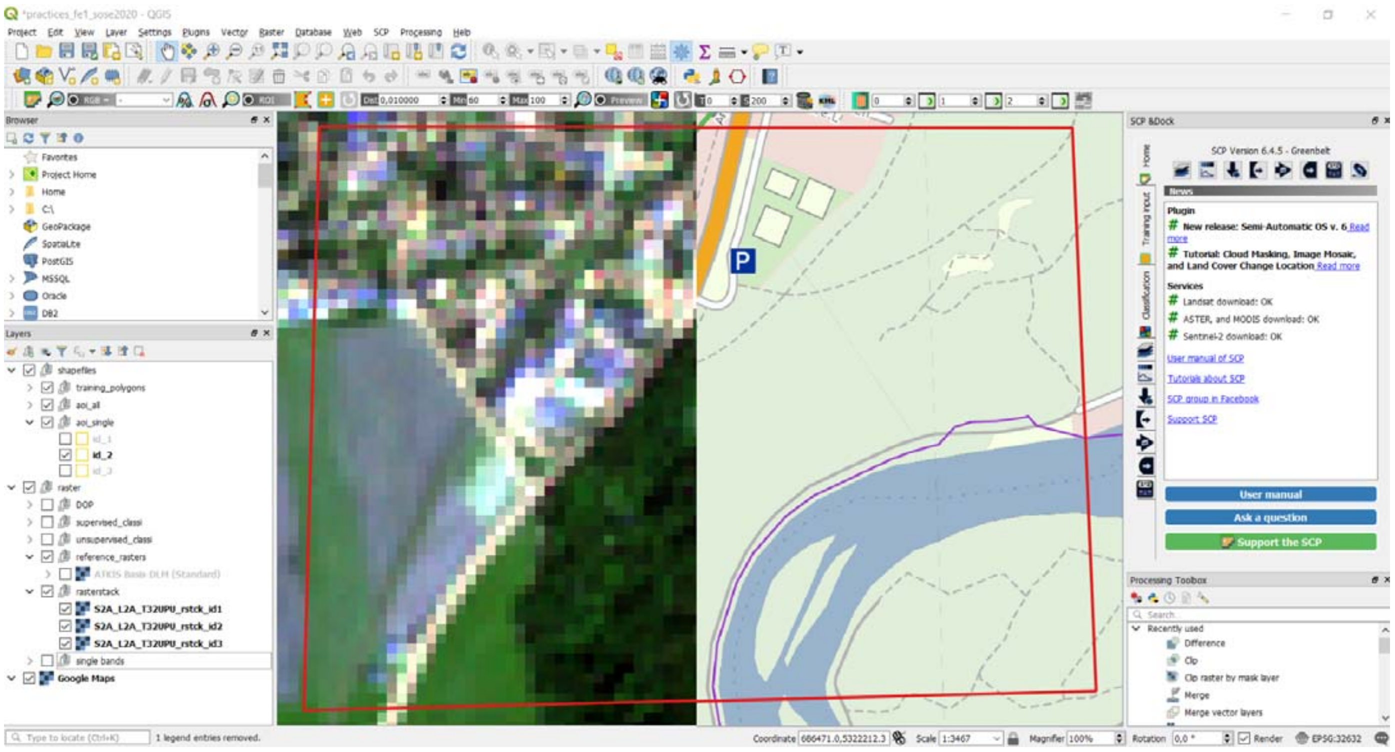
Feature limit for GetFeatureInfo: 10

ETRS89 / UTM zone 32N

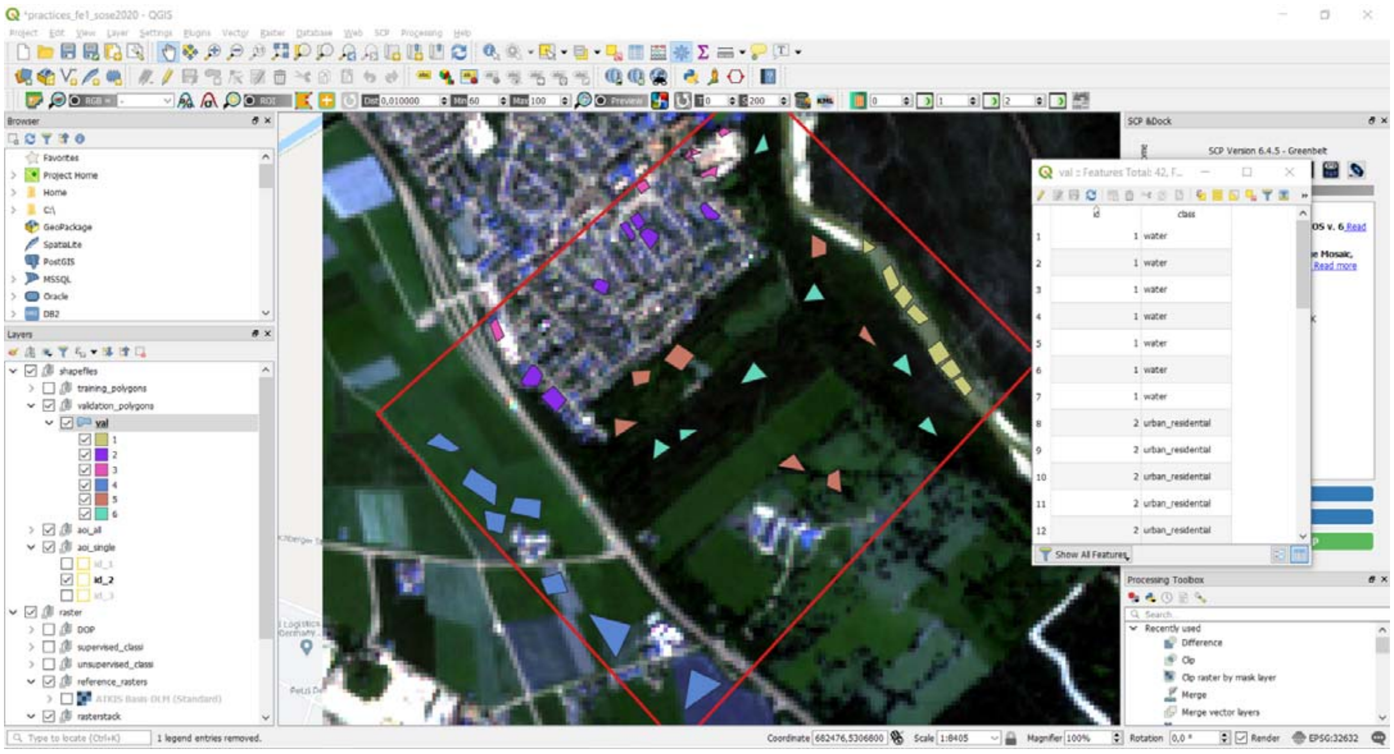
Layer name: ATKIS Basis-DLM (Standard)

1 Layer(s) selected

2. Compare all layers, using the reference data to create your polygons.



3. Create a new set of shapefiles called "validation.shp" and with the help of the reference data, draw again at least 15 polygons per landclass. NOTICE: you have to use the same number, id and label of landclasses that you used for your training polygons.



	id	class
1	1	water
2	1	water
3	1	water
4	1	water
5	1	water
6	1	water
7	1	water
8	2	urban_residential
9	2	urban_residential
10	2	urban_residential
11	2	urban_residential
12	2	urban_residential

B. In Python: set up

4. The first thing you always have to do, is to define which libraries or packages you will need. You don't need to change anything here. However, if you don't run this cell, the code won't work.

In [1]:

```
import os
import numpy as np # math and array handling
import matplotlib.pyplot as plt # plot figures
from matplotlib.colors import ListedColormap # for plotting, import color palettes
from osgeo import gdal, ogr, gdal_array # I/O validation data, rasters
import rasterio as rio # Rasterio reads and writes GEOTIFF data and makes it understandable
from skimage import exposure # to perforom histogram equalization of rasters
```

5. To do the Accuracy Assessment we are going to use some functions provided by the scikit-learn library

In [2]:

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import pandas as pd # to write and handle dataframes
```

6. To display our confusion matrix we are going to use the seaborn library which we are going to install using the anaconda prompt with the "!" operator:

In [3]:

```
!conda install seaborn -y
import seaborn as sns

# NOTICE: open Anaconda prompt or Command line and update conda if you get the following me
'''
==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.3

Please update conda by running

    $ conda update -n base -c defaults conda
'''
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

Out[3]:

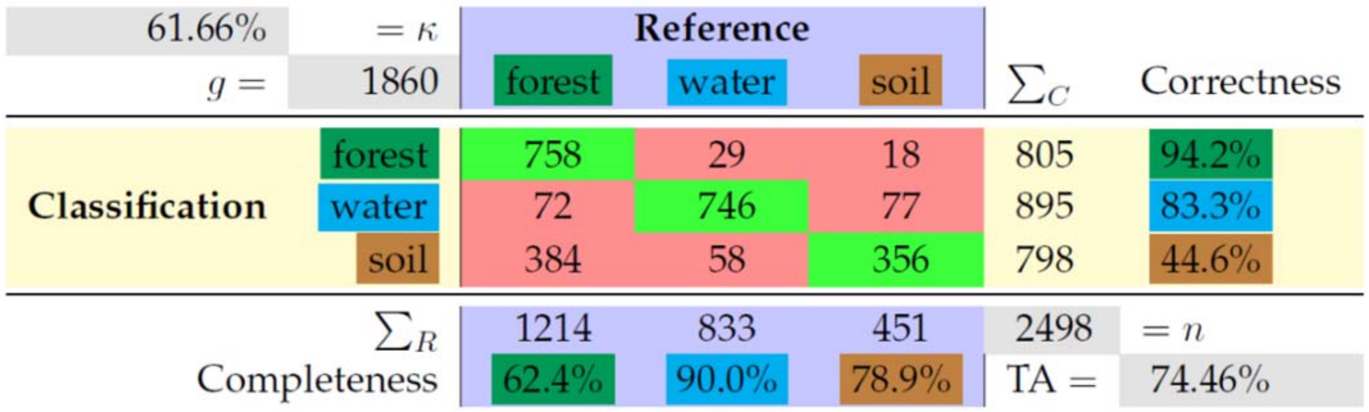
```
'\n==> WARNING: A newer version of conda exists. <==\n  current version: 4.
8.2\n  latest version: 4.8.3\n\nPlease update conda by running\n\n    $ cond
a update -n base -c defaults conda\n'
```

7. Afterwards, you have to define the paths where your data is located. You can also specify where you want the results to be saved. Please adapt the following code, with the path of the files on your computer. Remember to add an "r" at the beginning of your path.

In [4]:

```
# path where my data is located
folder_src = r"C:\Users\ulloa-to\PythonProjects\practices_fe1_ss2020\scripts\data\p6"

# path where I want to save my results
folder_results = r"C:\Users\ulloa-to\PythonProjects\practices_fe1_ss2020\scripts\data\p6\re
```

C. Accuracy Assessment (EXERCISE): Understanding Confusion Matrix and Error estimation



61.66%	$= \kappa$	Reference				
$g =$	1860	forest	water	soil	$\sum_C$	Correctness
Classification	forest	758	29	18	805	94.2%
	water	72	746	77	895	83.3%
	soil	384	58	356	798	44.6%
	$\sum_R$	1214	833	451	2498	$= n$
	Completeness	62.4%	90.0%	78.9%	TA $=$	74.46%

8. Let us make a hands-in exercise on Accuracy Assessment with the data from the lecture (Chapter "Validation" page 7). The idea is to calculate yourself the different errors Total Accuracy, Completeness, and Correctness (also known as Overall Accuracy, Producer's accuracy, and User's accuracy.)

More information on Errors' estimation (http://gis.humboldt.edu/OLM/Courses/GSP_216_Online/lesson6-2/metrics.html)

Create matrix of values with landclasses as labels

In [5]:

```
# create the labels for the classification file
index = ["forest", "water", "soil", "sum_rows"]

# create the labels for the reference data
columns = ["forest", "water", "soil", "sum_cols"]

# create some random values
values = np.array([[758, 29, 18, 805],
                  [72, 746, 77, 895],
                  [384, 58, 356, 798],
                  [1214,833,451,2498]]) # as an array

# turn the array into a pandas dataframe (table)
cm_df = pd.DataFrame(values, index=index) # read it in a table
cm_df.columns = columns # define which ones are the labels for the columns
print(cm_df) # show
```

```
          forest  water  soil  sum_cols
forest       758     29    18       805
water         72    746    77       895
soil         384     58   356       798
sum_rows    1214    833   451      2498
```

Total or Overall accuracy

In [6]:

```
# Total Accuracy / Overall accuracy = sum of the diagonal/total number of pixels

OA = (758 + 746 + 356) / 2498
OA
```

Out[6]:

```
0.7445956765412329
```

In [7]:

```
print('The Overall accuracy is', format(OA*100), '%')
print('The Overall accuracy is {:.2f}%'.format(OA*100)) # notice, print with point, not com
```

```
The Overall accuracy is 74.45956765412329 %
The Overall accuracy is 74.46%
```

Completeness or Producer's accuracy

In [8]:

```
# Producer's accuracy = correctly classified pixels / total number of reference sites

PA_forest = 758/1214
PA_water = 746/833
PA_soil = 356/451
```

In [9]:

```
print('Completeness or Producer accuracy Report')
print('The Producer accuracy for FOREST is {:.2f}%'.format(PA_forest*100))
print('The Producer accuracy for WATER is {:.2f}%'.format(PA_water*100))
print('The Producer accuracy for SOIL is {:.2f}%'.format(PA_soil*100))
```

```
Completeness or Producer accuracy Report
The Producer accuracy for FOREST is 62.44%
The Producer accuracy for WATER is 89.56%
The Producer accuracy for SOIL is 78.94%
```

Correctness or User's accuracy

In [10]:

```
# User's accuracy = correctly classified pixels / total number of classified sites

UA_forest = 758/805
UA_water = 746/895
UA_soil = 356/798
```

In [11]:

```
print('Correctness or User accuracy Report')
print('The User accuracy for FOREST is {:.2f}%'.format(UA_forest*100))
print('The User accuracy for WATER is {:.2f}%'.format(UA_water*100))
print('The User accuracy for SOIL is {:.2f}%'.format(UA_soil*100))
```

```
Correctness or User accuracy Report
The User accuracy for FOREST is 94.16%
The User accuracy for WATER is 83.35%
The User accuracy for SOIL is 44.61%
```

Kappa coeficient

In [32]:

```
# Calculate Kappa

# First, calculate K (equation 6 in the lecture)
K = (1214*805 + 833*895 + 451* 798)
# Second, list all the other variables needed
g = 758 + 746 + 356 # correctly classified pixels
n = 2498 # sum of all pixels
n2 = pow(n, 2) # n to the power of 2
# Third, use all variables to calculate kappa (equation 7 in the lecture)
kappa = (g*n-K)/(n2 - K)

print('The Kappa coeficient is {:.2f}'.format(kappa))
```

```
The Kappa coeficient is 0.62
```

9. We can play around with another data. In the next example, take a look at the following data.
   - y_true: are the real values of the pixels
   - y_pred: are the values predicted by the classification

In [33]:

```
y_true = [2, 0, 2, 2, 0, 1, 0, 2, 1, 2, 2, 1]
y_pred = [0, 0, 2, 2, 0, 2, 0, 1, 2, 2, 0, 1]
```

10. How are the classification classes named and how many are they?

In [ ]:

```
 
```

11. To which dataset "reference" and "classification", correspond "y_true" and "y_pred"?

In [ ]:

```
 
```

12. Using y_true, y_pred, build a confusion matrix, using the function 'confusion_matrix()'. Store the result in an object called 'confu_test'. For more info, visit: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

In [ ]:

```
 
```

13. Plot your object 'confu_test' using the following code:

```
plt.figure(figsize=(10,7))
sns.heatmap(confu_test, annot=True, fmt='g',cmap = 'YlGnBu')
plt.xlabel('classes - predicted')
plt.ylabel('classes - truth')
plt.show()
```

In [ ]:

```

```

14. Manipulate the arrays 'y_true' and 'y_pred' to increase the accuracy of class '1'. (hint: there are 2 possible things to do)

In [ ]:

```

```

15. Now we can run exactly the same analysis from exercise Nr.8, using a function in Python that summarizes all the previous calculations in one step. For this, we will use the same dataset, but change the labels of the landclasses to values:
    FROM ["forest", "water", "soil"] TO [1, 2, 3,]

In [14]:

```python
# define labels of rows and columns
index = [1, 2, 3, 'r_sum']
columns = [1, 2, 3, 'c_sum']
# fill in values
values = np.array([[758, 29, 18, 805],
                  [72, 746, 77, 895],
                  [384, 58, 356, 798],
                  [1214,833,451,2498]])

# turn the array into a pandas dataframe (table)
cm_df = pd.DataFrame(values, index=index)
cm_df.columns = columns
print(cm_df)
```

```
          1    2    3  c_sum
1       758   29   18    805
2        72  746   77    895
3       384   58  356    798
r_sum  1214  833  451   2498
```

Accuracy Assessment function on Python

In [15]:

```
# Let's configure the floating point precision of our output
np.set_printoptions(precision=2) # 2 decimals

# function for error estimations (AA)
def accAss(name, gcm, n_classes):
    #Completeness
    print('Accuracy Assessment for',name,'Prediction')
    comp = np.zeros(n_classes)
    for i in range(n_classes):
        comp[i]=gcm[i+1][i+1]/gcm[i+1]['r_sum']
    print('Producer Accuracy [%]:', (comp*100))

    #Correctness
    corr = np.zeros(n_classes)
    for i in range(n_classes):
        corr[i]=gcm[i+1][i+1]/gcm['c_sum'][i+1]
    print('User Accuracy [%]:',(corr*100))

    #Total Accuracy
    diag = np.zeros(n_classes)
    for i in range(n_classes):
        diag[i]=gcm[i+1][i+1]
    g = np.sum(diag)
    n = gcm['c_sum']['r_sum']
    ta = g/n
    print('Total Accuracy [%%]: %.2f' % (ta*100))

    #Correlation
    corr_v = np.zeros(n_classes)
    for i in range(n_classes):
        corr_v[i]=gcm[i+1]['r_sum']*gcm['c_sum'][i+1]
    K = np.sum(corr_v)

    #Cohen's Kappa
    k=(g*n-K)/(n*n-K)
    print("Cohens's Kappa: %.2f" % k)
```

16. Apply this function on the same example of the lecture (exercise Nr.15), and compare with the results you calculated by hand (exercise Nr.8):

In [16]:

```
# apply function for AA on table 'cm_df', with 3 classes
accAss('Lecture Example',cm_df, 3)
```

```
Accuracy Assessment for Lecture Example Prediction
Producer Accuracy [%]: [62.44 89.56 78.94]
User Accuracy [%]: [94.16 83.35 44.61]
Total Accuracy [%]: 74.46
Cohens's Kappa: 0.62
```

D. Load raster and vector data

17. On this section you will load your raster and vector data and perform an accuracy assessment of your

classification. The process of loading the data is exactly the same as for last week: classification.

Load Sentinel2-A raster

18. Load the raster file that you want to. Adapt the code to your data.

In [17]:

```
# path to the classified image (Setinel-2A Rasterstack)
s2_stack = os.path.join(folder_src, 'raster','S2A_L2A_T32UPU_rstck_id2.tif')
```

19. Extract the information of your raster data. Here, you make the attributes readable for the classification algorithm. Afterwards, the data is being transformed into a Numpy array for easier calculations. Check the printing commands and the variables that are being used!

In [18]:

```
# load image data
#In this script we are Using gdal.open() instead of rio.open()
img_ds = gdal.Open(s2_stack, gdal.GA_ReadOnly)

img = np.zeros((img_ds.RasterYSize, img_ds.RasterXSize, img_ds.RasterCount),
               gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBand(1).DataType))
for b in range(img.shape[2]):
    img[:, :, b] = img_ds.GetRasterBand(b + 1).ReadAsArray()

print("Raster format is:", gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBand(1)

# store the variables above in a more meaningful way. You will use these variables later.
row = img_ds.RasterYSize
col = img_ds.RasterXSize
band_number = img_ds.RasterCount

print("Raster number of rows: {}".format(row))
print("Raster number of columns: {}".format(col))
print("Raster number of bands: {}".format(band_number))
```

```
Raster format is: <class 'numpy.float32'>
Raster number of rows: 186
Raster number of columns: 209
Raster number of bands: 4
```

Load Minimum Distance and Random Forest classification rasters

20. Define individual paths for your classification rasters

In [19]:

```
# path to the classification rasters, depending of the algorithm used
#Random Forest
rf_pred_path = os.path.join(folder_src, 'raster', 'rf_nt250.tif')
#Minimum Distance
knn_pred_path = os.path.join(folder_src, 'raster', 'md_classi.tif')
```

21. Now we are going to load the prediction result rasters from the last practice and see if the dimensions are correct (they should match the extent of your Sentinel-2a image):

In [20]:

```
# Check dimensions of RF classification

# open file
rf_open = gdal.Open(rf_pred_path, gdal.GA_ReadOnly)
# extract metadata
rf_meta = np.zeros((rf_open.RasterYSize, rf_open.RasterXSize, rf_open.RasterCount), gdal_ar
for b in range(rf_meta.shape[2]):
    rf_meta[:, :, b] = rf_open.GetRasterBand(b + 1).ReadAsArray()
# define the new shape for the rf classification
new_shape = (rf_meta.shape[0], rf_meta.shape[1])
# reshape
rf_pred = rf_meta[:, :, :np.int(rf_meta.shape[2])].reshape(new_shape)

# compare the dimensions. They are the same as S2a
print("Dimensions of RF after 'reshape()' are",rf_pred.shape)
```

```
Dimensions of RF after 'reshape()' are (186, 209)
```

In [21]:

```
# Check dimensions of MD classification

# open file
knn_open = gdal.Open(knn_pred_path, gdal.GA_ReadOnly)
# extract metadata
knn_meta = np.zeros((knn_open.RasterYSize, knn_open.RasterXSize, knn_open.RasterCount), gda
for b in range(knn_meta.shape[2]):
    knn_meta[:, :, b] = knn_open.GetRasterBand(b + 1).ReadAsArray()
# define the new shape for the md classification
new_shapeknn = (knn_meta.shape[0], knn_meta.shape[1])
# reshape
knn_pred = knn_meta[:, :, :np.int(knn_meta.shape[2])].reshape(new_shapeknn)

# compare the dimensions. They are the same as S2a
print("Dimensions of MD after 'reshape()' are",knn_pred.shape)
```

```
Dimensions of MD after 'reshape()' are (186, 209)
```

Load validation shapefile

22. Load your validation data (shapefiles). Define the column that has the attributes (i.e. 'landclass' OR 'class' OR 'landcover'). Adapt the code to your data.

In [22]:

```
#validation as shape files
validation = os.path.join(folder_src,'vector','val.shp')
```

In [23]:

```
# what is the attributes name of your classes in the shape file (field name of the classes)
attribute = 'id'
```

23. Recheck if the validation data is in the correct format:

In [24]:

```
# load validation data and show all shape attributes
shape_dataset = ogr.Open(validation)
shape_layer = shape_dataset.GetLayer()

# extract the names of all attributes (fieldnames) in the shape file
attributes = []
ldefn = shape_layer.GetLayerDefn()
for n in range(ldefn.GetFieldCount()):
    fdefn = ldefn.GetFieldDefn(n)
    attributes.append(fdefn.name)

# print the attributes
print('Available attributes in the shape file are: {}'.format(attributes))
```

```
Available attributes in the shape file are: ['id', 'class']
```

24. Rasterize your validation polygons

In [25]:

```
# load training data from shape file and rasterize

shape_dataset = ogr.Open(validation)
shape_layer = shape_dataset.GetLayer()

mem_drv = gdal.GetDriverByName('MEM')
mem_raster = mem_drv.Create('',img_ds.RasterXSize,img_ds.RasterYSize,1,gdal.GDT_UInt16)
mem_raster.SetProjection(img_ds.GetProjection())
mem_raster.SetGeoTransform(img_ds.GetGeoTransform())
mem_band = mem_raster.GetRasterBand(1)
mem_band.Fill(0)
mem_band.SetNoDataValue(0)

att_ = 'ATTRIBUTE='+attribute
# http://gdal.org/gdal__alg_8h.html#adfe5e5d287d6c184aab03acbfa567cb1
# http://gis.stackexchange.com/questions/31568/gdal-rasterizelayer-doesnt-burn-all-polygons
err = gdal.RasterizeLayer(mem_raster, [1], shape_layer, None, None, [1],  [att_,"ALL_TOUCHE
assert err == gdal.CE_None

val = mem_raster.ReadAsArray()
```

25. Voilà! Now you have your validation polygons in the same resolution and projection as your Sentinel-2 Rasterstack. The next task is to extract the validation pixels (OR 'samples') from the Rasterstack. The samples will be used for measuring the accuracy of the classification raster. In total, I am extracting 1091 samples from the matrix 'X'.

In [26]:

```
# Number of validation pixels:
n_samples = (val > 0).sum()
print('{n} training samples'.format(n=n_samples))

# What are our classification labels?
labels = np.unique(val[val > 0])
print('training data include {n} classes: {classes}'.format(n=labels.size, classes=labels))

# Subset the image dataset with the image = X
# Mask the classes on the validation dataset = y
# These will have n_samples rows
X_rf = rf_pred[val > 0]
y = val[val > 0]

print('Our X matrix is sized: {sz}'.format(sz=X_rf.shape))
print('Our y array is sized: {sz}'.format(sz=y.shape))
```

```
1091 training samples
training data include 6 classes: [1 2 3 4 5 6]
Our X matrix is sized: (1091,)
Our y array is sized: (1091,)
```

26. Why the dimensions of both X and y are the same? How many bands each object has? Compare this result with the step Nr.10 from Practice 5.

In [ ]:

```

```

27. What determines the number of samples from the polygons?

In [ ]:

```

```

Visualization of data

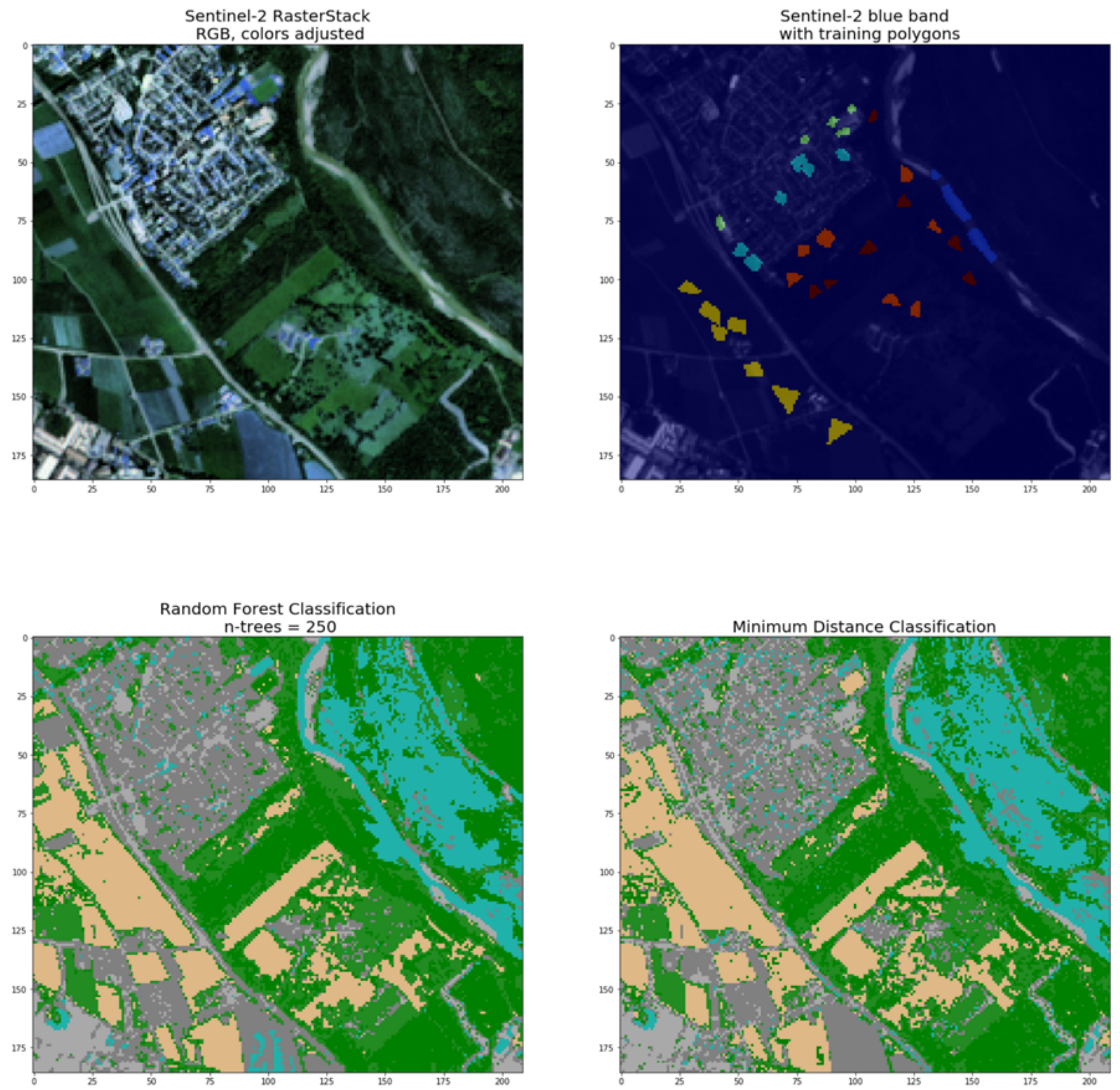
28. Plot all your files: S2A rasterstack, validation polygons, classification rasters

In [27]:

```
# Define color palette
#[water, urban residential, urban industrial, agricultural, broadleaved forest, coniferous
custom_cmap = ListedColormap(["lightseagreen","grey","darkgrey","burlywood", "forestgreen",
```

In [28]:

```
# Define the size of the graphs
#plot=plt.subplots(figsize=(20,20))
fig,ax = plt.subplots(figsize=(20,20))
# Define the spacing among graphs
fig.tight_layout(pad=4.0)

# Plot the rasterstack with an Adaptive Equalization
plt.subplot(221)
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
plot_rs = plt.imshow(img_adapteq)
plt.title('Sentinel-2 RasterStack\n RGB, colors adjusted', fontsize=20)
## More info: https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.

# Plot the blue band with the training polygons
plt.subplot(222)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
# Add your training polygons on top. NOTICE that you have here validation and not training
plot_tr = plt.imshow(val, cmap='jet', alpha=0.5)
plt.title("Sentinel-2 blue band\n  with training polygons", fontsize=20)

# Plot the RandomForest classification
plt.subplot(223)
plot_rf = plt.imshow(rf_pred, cmap=custom_cmap)
plt.title('Random Forest Classification\n n-trees = 250', fontsize=20)
## I eliminated here the colorbar, for visualization purposes

# Plot the Minimum Distance classification
plt.subplot(224)
plot_md = plt.imshow(knn_pred, cmap=custom_cmap)
plt.title('Minimum Distance Classification', fontsize=20)
## I eliminated here the colorbar, for visualization purposes

# show your plots. This command has to be run only once.
plt.show()
```

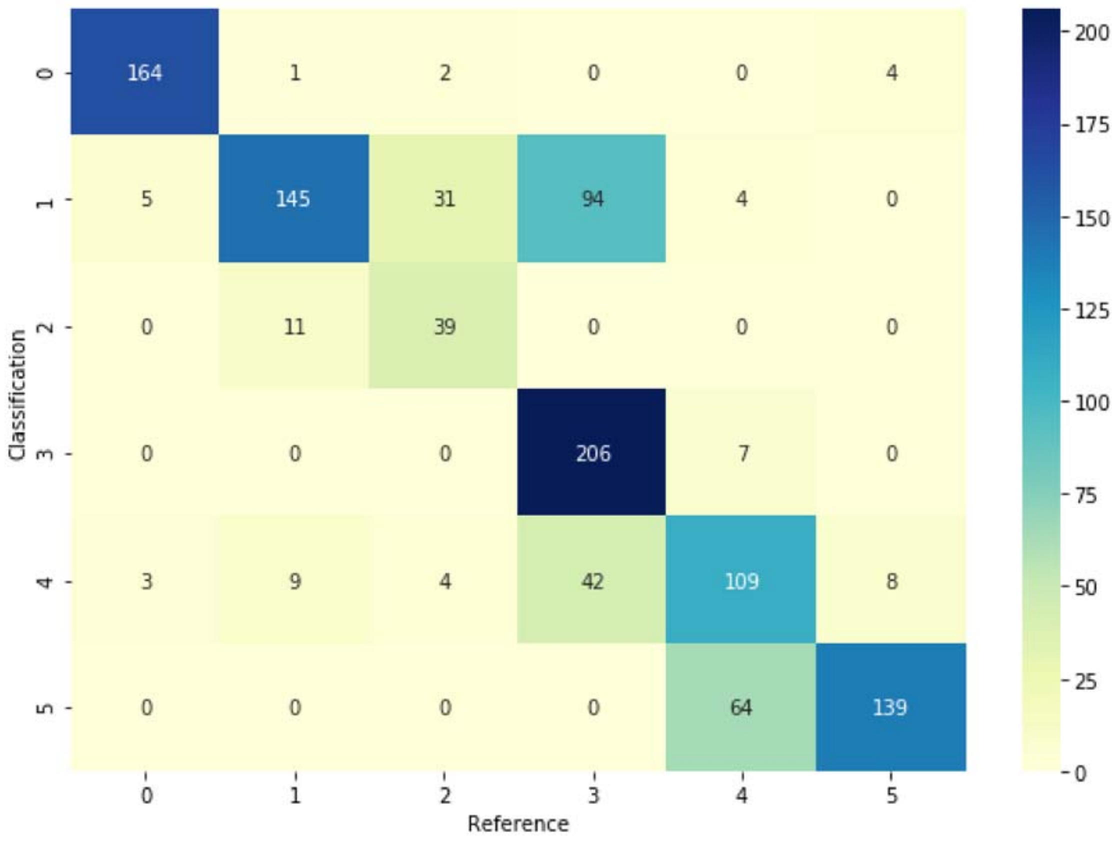


E. Accuracy Assessment: Estimate Confusion Matrix and Errors on your classification

29. Plot the confusion matrix of your Random Forest classification

In [29]:

```
# use the function 'confusion_matrix' from scikit-learn to plot the confusion matrix of you
cm_val = confusion_matrix(X_rf, y)

# define size of plot
plt.figure(figsize=(10,7))
# color the table. The higher the value, the darker the color
sns.heatmap(cm_val, annot=True, fmt='g',cmap = 'YlGnBu' )
# labels
plt.xlabel('Reference')
plt.ylabel('Classification')
# plot
plt.show()
# print text
print('Confusion matrix of Random Forest classification\n Landclasses:\n 0 = water\n 1 = ur
```



Classification \ Reference	0	1	2	3	4	5
0	164	1	2	0	0	4
1	5	145	31	94	4	0
2	0	11	39	0	0	0
3	0	0	0	206	7	0
4	3	9	4	42	109	8
5	0	0	0	0	64	139

```
Confusion matrix of Random Forest classification
 Landclasses:
 0 = water
 1 = urban residential
 2 = urban industrial
 3 = agriculture
 4 = deciduous forest
 5 = coniferous forest
```

30. Drop your results in a pandas table

In [30]:

```
# Compute a simple cross tabulation of two (or more) factors. By default computes a frequen
rcm = pd.crosstab(X_rf, y, margins=True)

# rename the colums and rows of the table
rcm.rename(columns={'All':'c_sum'}, index={'All':'r_sum'}, inplace=True)
print(rcm)
```

```
col_0    1    2   3    4    5    6  c_sum
row_0
1      164    1   2    0    0    4    171
2        5  145  31   94    4    0    279
3        0   11  39    0    0    0     50
4        0    0   0  206    7    0    213
5        3    9   4   42  109    8    175
6        0    0   0    0   64  139    203
r_sum  172  166  76  342  184  151   1091
```

31. Apply the Acuracy Assessment function on the confusion matrix of your Random Forest classification

In [31]:

```
# apply function for AA on table 'rcm', with 6 classes
accAss('Random Forest',rcm, 6)
```

```
Accuracy Assessment for Random Forest Prediction
Producer Accuracy [%]: [95.35 87.35 51.32 60.23 59.24 92.05]
User Accuracy [%]: [95.91 51.97 78.   96.71 62.29 68.47]
Total Accuracy [%]: 73.51
Cohens's Kappa: 0.68
```

32. Verify 'by hand' that the errors estimation of the RF classification is correct

In [ ]:

33. Calculate the confusion matrix and estimate the errors of your Minimum Distance classification.

In [ ]:

```
```

34. Plot the confusion matrix of your MD classification

In [ ]:

```
```

35. Is the any difference in the AA of every classification? What are the reasons for this?

In [ ]:

```
```

This tutorial was prepared with the support from Gabriel Cevallos. June 2020