Landcover classification with ArcGIS Pro:

Using the tool *Mosaic To New Raster* prepare the orthophotographs in folder VilniusOrtho to match the requirements of ArcGIS Living Atlas of the world deep learning model – High Resolution Land Cover Classification – USA (package).

Note: For the DL packages to work in ArcGIS Pro, it is required to have installed the ArcGIS Pro Deep Learning Libraries (GitHub library installer download) and also have additional extensions, which the models rely on (e. g. Image Analyst, 3D Analyst).

After transforming your rasters to an appropriate resolution, landcover classification is set using *Classify Pixels Using Deep Learning*. It is recommended to use GPU in the environments tab, other than that the parameters are available on the guide page of the DL package.

__________________ Alternative with QGIS (less advanced):

Download the Semi-Automatic Classification Plugin for QGIS. Create a signature file and mark your class features on the raster you're using, calculate the spectral signatures if the box automatically calculating the signatures is not checked (recommended: after finishing creating a class, select all features and right click to calculate signatures). Then, set the classification algorithm in the opened plugin to Maximum Likelihood Estimate and let the algorithm fill the possible holes (check box). This is a classic way of classifying pixels. To get better results it is important to have enough representative samples and raster resolution of ≥ 50 cm. __________________

City block generation using Python notebook (Jupyter Notebook):

First, create a new folder for your project, preferably in the C:\ location (without a long path).

Go to this GitHub page: DaVaiva and clone the repository of your choice to your computer as so:

Click on: <> Code -> Download ZIP. And unpack it in your new folder (C:\your_project_folder), where you will be keeping everything that is connected to your project.

___________________________________ Alternative with Git:

If you wish to clone repositories from github with HTTPS or GitHub CLI and overall tie your code workspace with GitHub to push and pull requests, you need to have Git installed on your computer (How-To).

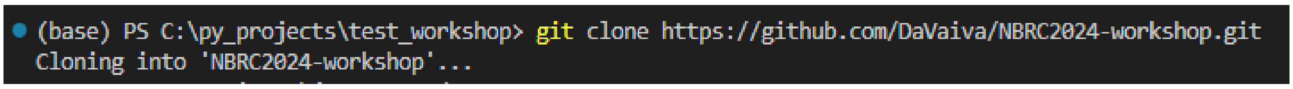
Go to this GitHub page: DaVaiva and clone the repository to your computer as so:

Go to a repository of your point of interest.

Click on: <> code -> copy URL to clipboard (HTTP method).

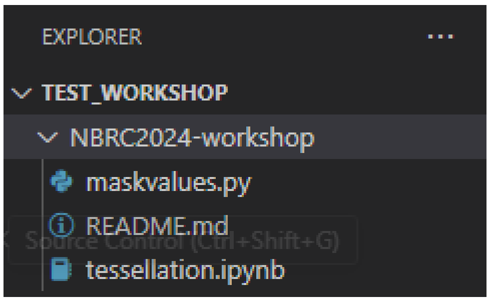
Open a new Visual Studio Code window and in the explorer tab, click *Clone Repository*. Select your newly created project folder (C:\your_project_folder). All files from the repository will be cloned to this folder.

Or type git clone and paste the link like this in the terminal when you have your project folder (e. g. test_workshop) already open in Visual Studio Code:



```
● (base) PS C:\py_projects\test_workshop> git clone https://github.com/DaVaiva/NBRC2024-workshop.git
  Cloning into 'NBRC2024-workshop'...
```

The explorer now has cloned the repository, creating a folder for it inside the project folder:



```
EXPLORER                    ···
∨ TEST_WORKSHOP
  ∨ NBRC2024-workshop
      maskvalues.py
      README.md
      tessellation.ipynb
```

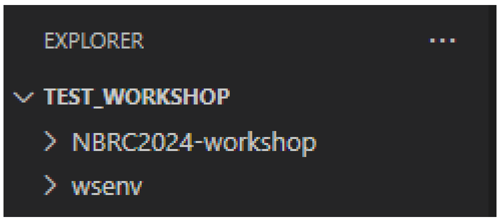
To keep the computer clean and versions in check, create a venv virtual environment (venvs can also be created with Anaconda and are considered conda environments. They are more comfortable to activate). A virtual environment is a way to keep different versions of libraries available for different projects because their functionality can change – new modules might be added, but some might be removed, the names can also change from version to version.

To create a virtual environment, you must have your_project_folder opened in VSC (Visual Studio Code) explorer, which -a reminder- should also hold your cloned GitHub repository.

Open a new terminal: Terminal -> new terminal (or Ctrl+Shift+`)

In the terminal, type: python -m venv yourenvironmentname (best to keep the name short).

You will see your new environment in your project folder:



```
EXPLORER                    ···
∨ TEST_WORKSHOP
  > NBRC2024-workshop
  > wsenv
```

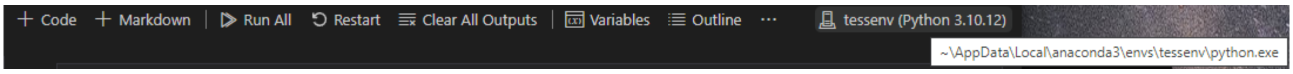
After creating an environment, it needs to be activated. In the terminal, copy the file path to the Scripts of your virtual environment folder e.g. C:\py_projects\test_workshop\wsenv\Scripts and add \Activate.ps1 (PowerShell terminal)

If Activate.ps1 does not work, add \activate.bat (cmd.exe terminal) to the end of the path. You should be able to see that the start of the file path in the terminal has changed from (base) to (myenvironment).

Install all the required libraries through the terminal: pip install libraryname... which are being imported at the top of the tessellation.ipynb code (you can view it in a tab by simply clicking on it). You can install multiple libraries at once just by typing and separating by a space as so: pip install matplotlib geopandas. Python installs more than one library sometimes, because they normally have dependencies.

Note: you can check (in terminal) if the library is present with pip show libraryname.

The code is set up in a way where you don't need to change much. When you will try to run your code in an interactive window, VSC (Visual Studio Code) will tell you, that you need to install ipykernel – install it. It is tied to your environment.



Kernels are used for interactive computing. Manual installation like this is required when we want to use a different version of Python, or a virtual environment or conda environment. When you have more projects made with different environments, you must select the right kernel when you open the project again (if it's not memorised), or else you will have errors for missing modules.

The code cells are prepared for execution, but minor changes have to be made. You need to change the data file paths to fit yours when you load your input and save your output data. Keep in mind the workshop files have errors on purpose as to teach how to deal with them.

________________________________ Alternative with GIS:

To create Voronoi polygons around other polygons in GIS desktop, you would need to:

Densify every polygon's vertices (Pro: *Densify*. QGIS: *Densify by interval*);

Turn the vertices into points (Pro: *Vertices To Points*. QGIS: *Extract vertices*);

Create Thiessen Polygons (for Pro, or *Voronoi polygons* for QGIS) and then *dissolve* them by original uid, that you had to carry throughout the processes, and you can't even add barriers.

You waste time on setting parameters and output locations, executing one process after another and taking up precious memory space with intermediate data.

____

The last thing to do before moving on to landcover analysis is to create gaps around city blocks/enclosures. To do it the "right" way, we would look at this document: D1-933 Dėl statybos techninio reglamento STR 2.06.04:2014 „Gatvės ir vietinės reikšmės keliai. Bendrieji... (lrs.lt), which states the width of road lanes in different category roads.

CHOICE.

- ➔ Ideally, we would create buffers for roads A1 to D2 and erase them from our blocks. This is standard data management and cleaning.
- ➔ For the purpose of moving on faster, we can create inward buffers around our blocks to separate them (*buffer*).

Landcover (percentage) attribute extraction

For this we will use a different python file from the GitHub workshop – maskvalues.py.

Theory. This is a python script, which executes everything when you run it. Scripts are cleaner than notebooks in the sense that it can be quite easy to get lost in your notebooks when you're testing out your code and its variations, possibilities, essentially analysing your outputs until you have a result that satisfies you and you've rid of all the errors. Scripts are used for automatization.

Of course, VSC offers debugging, but this is the approach I use as a fresh python user. After I decide that my code is finished, I tend to transfer it from a notebook to a script if it seems like a good idea for the use case.

This code calculates landcover class percentages in each block. The code has a few new libraries that need to be installed: rasterio and rasterstats. You can do it in the same environment. Open the terminal and pip install the libraries only when you make sure your environment is active. After installing, again, change all file paths before running the code in an interactive window. This can be quite an exhaustive process, so we will only do it on small subsets (QuickTest_Data\classified_ortho_subset.tif and encl_subset.shp) to see how it works and we will use the full output data later.

-ArcGIS Pro Model Builder challenges demonstration in the meantime-

Functional zone analysis

In the folder \QuickTest_Data you can find blocks_attributed.geojson. This file contains city blocks that have not only erased road categories, but bigger transport corridors according to the general plan and greenery layers. It does not cover the whole city, because it was labelled according to the attached landuse_types.jpg.

After calculating the percentages, we can export a table, which summarizes the mean landcover attributes for each of the functional zone classes (I've done it with ArcGIS Pro *Summary Statistics*, because I could not find a working similar base-QGIS implementation). They will be used to create custom rules later by interpretating the results. You can add extra fields such as adding up all impervious surfaces into one (Structures + Impervious + Impervio_1), calculate a coverage index (Structures / (Sum of all categories)) etc.

Load blocks_attributed.geojson into your QGIS project and use *Statistics by Categories* tool (if it works, or *summary statistics* in Pro if exported into .shp), where FZ attribute is the case field. Export table to excel (QGIS: *Export to spreadsheet*, ArcGIS Pro: *Table to Excel*). Do create a FZ_predicted text/string (5 symbol length) field for blocks_attributed.geojson.

You can find an exported excel table blocks_att_summary.xls in \QuickTest_Data folder. Try and create expressions (e. g. Imperv_all >= 70 & Structures >= 50) for each FZ category using the summarised fields. Highest and lowest values are coloured for easier interpretation.

Functional Zone rule implementation with ipynb

When you're mostly happy with your logical expressions. Open ClassifyCityBlocks.ipynb in VSC. You can create a copy of the file and copy cells one by one, especially where you can find the conditions. Copying one condition is enough as to not get overwhelmed what you need to change in each row. You can copy the first condition and change it to the next one until you reach the end. What's more, you don't have to load all your fields if you've not used them for your expressions. You can make these changes for the gdf_columns variable before loading blocks_attributed.geojson into memory as your geodataframe (gdf).

You can view how the blocks have "classified" by plotting them according to the FZ_predicted attribute that is being updated with logical expressions.

Note. DO NOT abuse the expressions code cell. If ran once, and the data has not been loaded once more when you want to run it again, it will only update the cells that have not been classified, therefore it gives a false notion, that all blocks are filling in with categories.
.isna() checks if the fields have values as to not update them by a consequent expression. Due to this, when you want to run your conditions again, you first need to load your data into the computer's memory, so that the field would be empty. If you don't care if the field updates by a different expression, you can remove .isna().

Theory. You might have thought why not make the rules and create a loop, which would execute row by row? You CAN do this. This code uses fields to reach this goal. Why?

When searching for specific rows based on certain criteria (e.g., filtering or querying), the computer needs to locate the memory address of the starting point for each row. Since the rows are not necessarily contiguous, the computer has to find the memory locations of the fields for each row, which might not be as cache-friendly as operating on contiguous memory. Columns on the other hand are stored together contiguously in many implementations. The efficiency of searching or filtering operations in a Data Frame can be influenced by factors such as how the data is laid out in memory, the size of the DataFrame, and the specific operations being performed. Pandas .loc is used for accessing multiple columns and slicing the DataFrame.

----- Optional if we'll have time -----

Compare your rule-based functional zones with QGIS algorithms

True functional zone layer

First, you can load actual functional zones into your project as an ArcGIS REST server: https://gis.vplanas.lt/arcgis/rest/services/Interaktyvus_zemelapis/Bendrasis_planas_2021/MapServer. The server contains a lot of layers. Functional zones are located in Pagrindinio brėžinio sprendimai -> Funkcinės zonos.

-Demonstration how to get to an available service through web developer tools-

Now you can view how the functional zones should look like after the general PLAN is executed. The deviations can be indicative of the CURRENT situation yet can also have errors. The fact is – suspicious territories can be identified and inspected quicker. These results can direct the attention to possible deviations and other developments. The rules can be shared with urban experts, who could give suggestions for any changes/corrections if they wish.

To compare your results with QGIS algorithms, find blocks_train.geojson and blocks_val.geojson in the /QuickTest_Data folder and load them into your project.

Theory. blocks_attributed.geojson has been split into training and validation sets using Python. Since the amount of data was not big, especially in some classes, the data was split traditionally 80% & 20% respectfully with the additional request that the split would consider the classes – all types of samples would end up in the training set. If you split data completely at random, it might have made the classes more imbalanced and perhaps left one or two behind. This is also a reason why machine learning was not used for this use case.

Orfeo Toolbox

Using QGIS, we will conduct an object-based classification. For this you will need the Orfeo Toolbox (OTB) plugin. Depending on each case, you might need to download it in a QGIS plugins folder (C:\QGIS-plugins) and refer to OTB application folder (e. g. C:/QGIS-plugins/OTB-8.0.1-Win64/lib/otb/applications) and OTB folder (C:/QGIS-plugins/OTB-8.0.1-Win64) in QGIS Processing settings next to Providers. Then restart your project. It's a powerful QGIS plugin for remote sensing. Object-based image classifications are available through this plugin. Right now, we will only use the last operations: *TrainVectorClassifier* and *VectorClassifier*.

OTB algorithms

Select the tool *TrainVectorClassifier*:

Input Vector Data: blocks_train.

Field names for training features – all landcover and additional calculated fields.

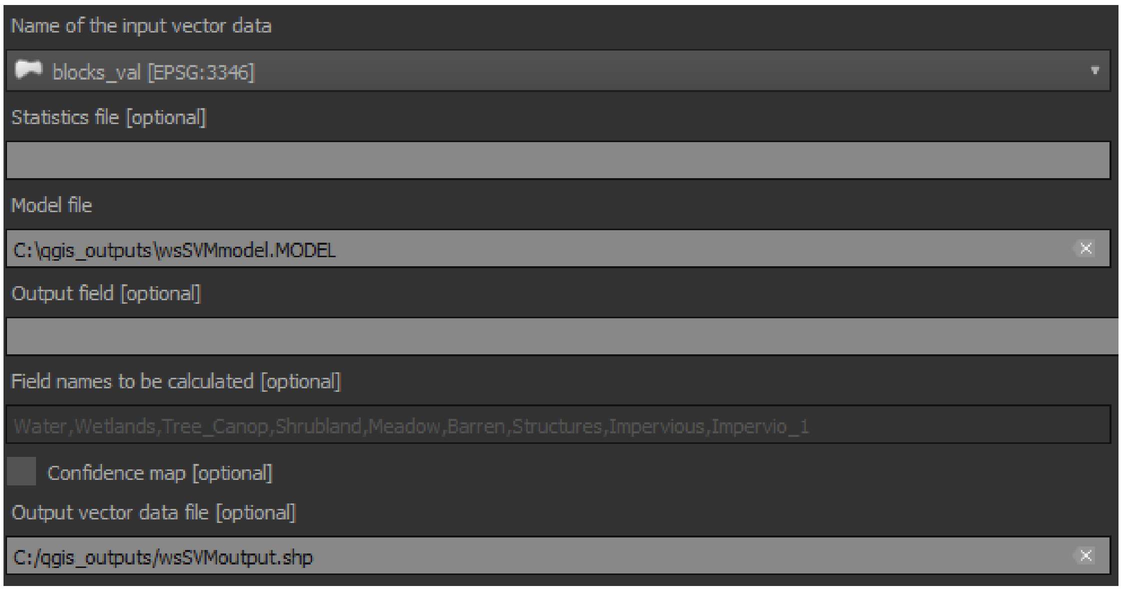
Field containing the class integer label for supervision: class_id.

Classifier to use for training – change to any of them (recommended to try libsvm, dt).

Theory. Random seed – whenever you see this parameter, it is there for reproducibility. Using this parameter makes sure that anyone who re-runs your code will get the exact same outputs. An example would be if you have a code that uses random number generating and you want someone else to run it, it will generate the same output while other repetitions will not match.

Save your output with the filetype .MODEL (e. g. svm_model.model). It will take a while.

When the model is created, select *VectorClassifier* from the processing toolbox:



Name of the input vector data
blocks_val [EPSG:3346]
Statistics file [optional]
Model file
C:\qgis_outputs\wsSVMmodel.MODEL
Output field [optional]
Field names to be calculated [optional]
Water,Wetlands,Tree_Canop,Shrubland,Meadow,Barren,Structures,Impervious,Impervio_1
Confidence map [optional]
Output vector data file [optional]
C:/qgis_outputs/wsSVMoutput.shp

Set input data to the validation set, load the model, select the same fields, and create an output.

Note. True machine learning requires a lot of testing and searching for the right hyperparameters (parameters that are set before running the process). It requires data science knowledge to select the right algorithms, although even experienced ML specialists do some different algorithm testing, they also examine data distributions, check for multicollinearity and so on...

Theory. You will probably like the Decision Tree (dt) algorithm here. One of the things I learned from data science, is that decision trees ALWAYS overfit (statistical model fits exactly against its training data). That is why it's good to access the inside of an algorithm to prune it, so that the model generalizes better. The decision boundary can get way too complicated, and the trees can have excessive branches. If you ever see a Random Forest algorithm – it is much better, Extra Trees algorithm – even better than RF.

Models are also extremely sensitive. What I mean by that is that there are various validation techniques and as you're perfecting a model, you have to keep generating different subsets of data as to not constantly train or validate the same model on the same data, it learns the distributions and can start to overfit. A great example of cross validation is the K-Fold technique. Whenever you validate only once, you might get a better result because you had a favourable subset, and the results could seem more positive than they actually are.

---Feel free to contact me with any questions 😊

Linkedin: Vaiva Venckauskaitė | LinkedIn

Email: vencvaiva@gmail.com