

Random Forest in remote sensing

GDP - Luca Ferrari, 2021



Index

- Random Forest
 - The model
 - Remote sensing applications
- Random Forest applications examples
 - Sentinel-2 + Copernicus CORINE Land Cover (Python)
 - Sentinel-2 + Custom classes (JS on GEE platform)

Random Forest

The model

The model

- RF is an ensemble classifier where classifications are made by combining the output of multiple decision trees
- The model has become popular in the Remote Sensing world due to the accuracy of the classifications and due to the multidimensional data handling capability

Random Forest Training (theory)

- A bootstrapped dataset is created from an original dataset and then used to train a decision tree with a subset of features at each step
- The procedure is repeated $nTree$ times so that a number of decision trees are trained on multiple bootstrapped datasets

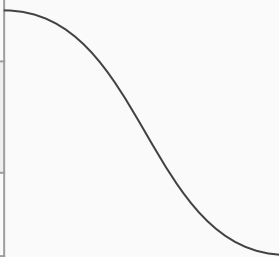
Random Forest Training (theory)

Original Dataset

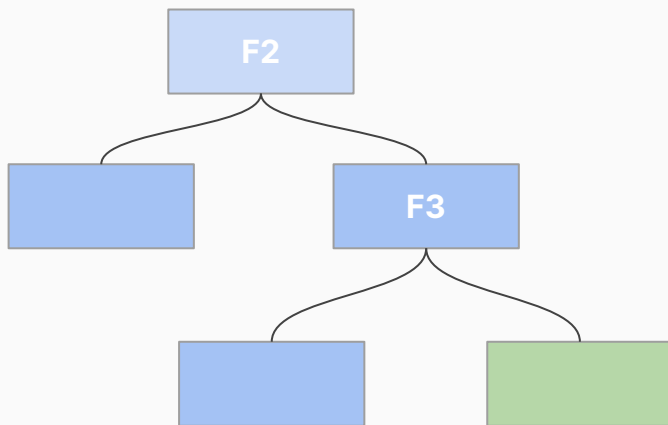
F1	F2	F3	CLASS
1	2	3	A
5	2	5	B
3	3	2	C
1	2	5	B

Bootstrapped Dataset

F1	F2	F3	CLASS
1	2	3	A
5	2	5	B
1	2	3	A



Random Forest Training (theory)

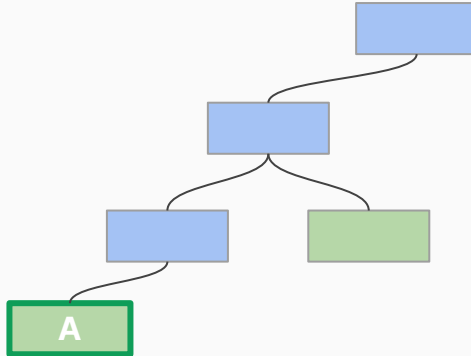
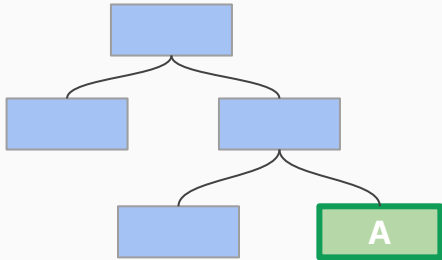


Bootstrapped Dataset

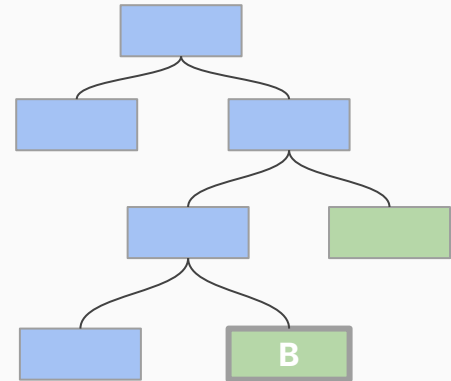
F1	F2	F3	CLASS
1	2	3	A
5	2	5	B
1	2	3	A
1	2	5	B

Random Forest Classification

F1	F2	F3	CLASS
1	2	3	A



A	B
2	1



Random Forest performance evaluation

- To evaluate the RF's accuracy we can exploit two different methods:
 - Out Of Bag validation set
 - external validation set
- nTree is the number of trees used in the RF
- mTry is the number of subset features used to train the decision trees

Remote sensing applications

Random Forest: Remote Sensing applications

- Random Forest has been used in different applications related to remote sensing:
 - Land Cover classes mapping
 - Urban buildings detection
 - Urban object classification from airborne LiDAR data

Random Forest: Pros and cons

- + Doesn't assume Normal distribution
- + Doesn't overfit
- + Correct for decision tree variance (by ensembling)
- + Not sensitive to outliers
- Memory intensive → Hyperspectral images problem → Rotation forests

Random Forest: Sensitivity to data

- Training and validation sets must be statistically independent
- Training samples must be class balanced
 - Unbalanced dataset problem and strength
- Spatial autocorrelation between TrSet and TsSet
- Training set dimension

Random Forest applications

First approach: Sentinel-2 + CORINE LC

First approach: Sentinel-2 + CORINE LC

- As a first approach I decided to use two different data sources to tackle the classification problem
- The idea is to use an already classified raster image as a dataset and then sample points from it to create a new training set for the RF model
- All the operations were carried out in Python interfacing with GEE through a library called *geemap*

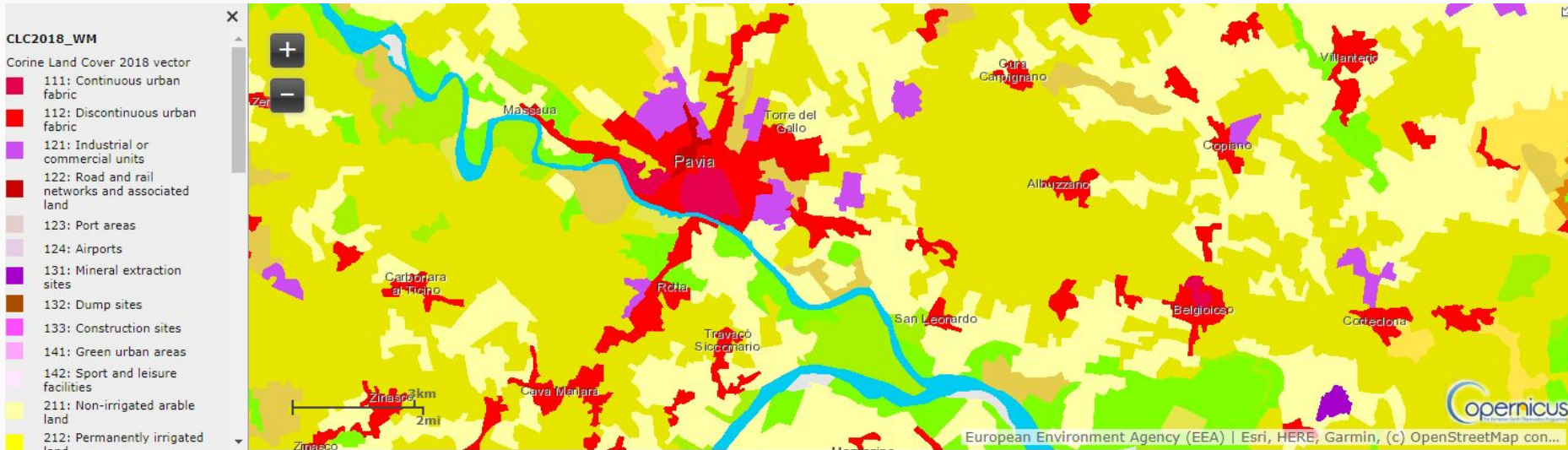
First approach: Sentinel-2 + CORINE LC

- + Accurate classes (specific and verified labels)
- + Flexible control on the number of samples
- + Easy model evaluation
- Scarse control on the sampling zones

First approach: Sentinel-2 + CORINE LC

- Satellite data comes from the Sentinel-2 satellite program
- Sentinel-2 was chosen because the spatial resolution of images was suitable for the application (20m)
- Classification data comes from the Copernicus CORINE Land Cover which provides a standardized data collection on land in Europe
- Data from 2017 was taken into account so that both the Sentinel-2 and CORINE Land Cover images were referring to the same exact time period

Copernicus CORINE Land Cover



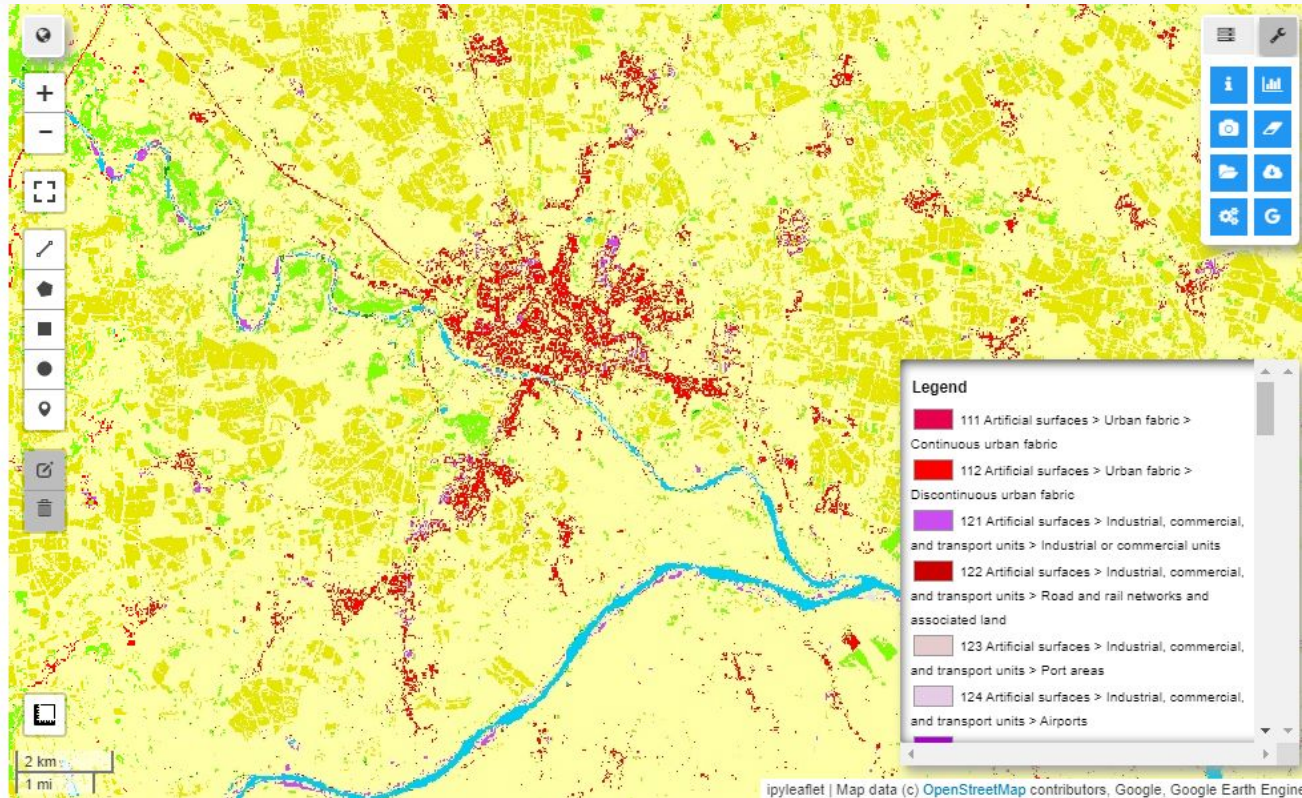
Data exploration and TrSet creation

- Data exploration was performed to verify that the image was suitable for being used as a proper dataset (cloud cover, basic corrections)
- 10k points were sampled from the Corine dataset and then used as regions in the Sentinel-2 satellite image to create a Training set for the RF model

Random Forest Training

- As a first experiment the RF model was trained with 10k samples points, using 100 decision trees and the default number of variables per split (\sqrt{p})
- The classification is then plotted on top of the map in a new layer

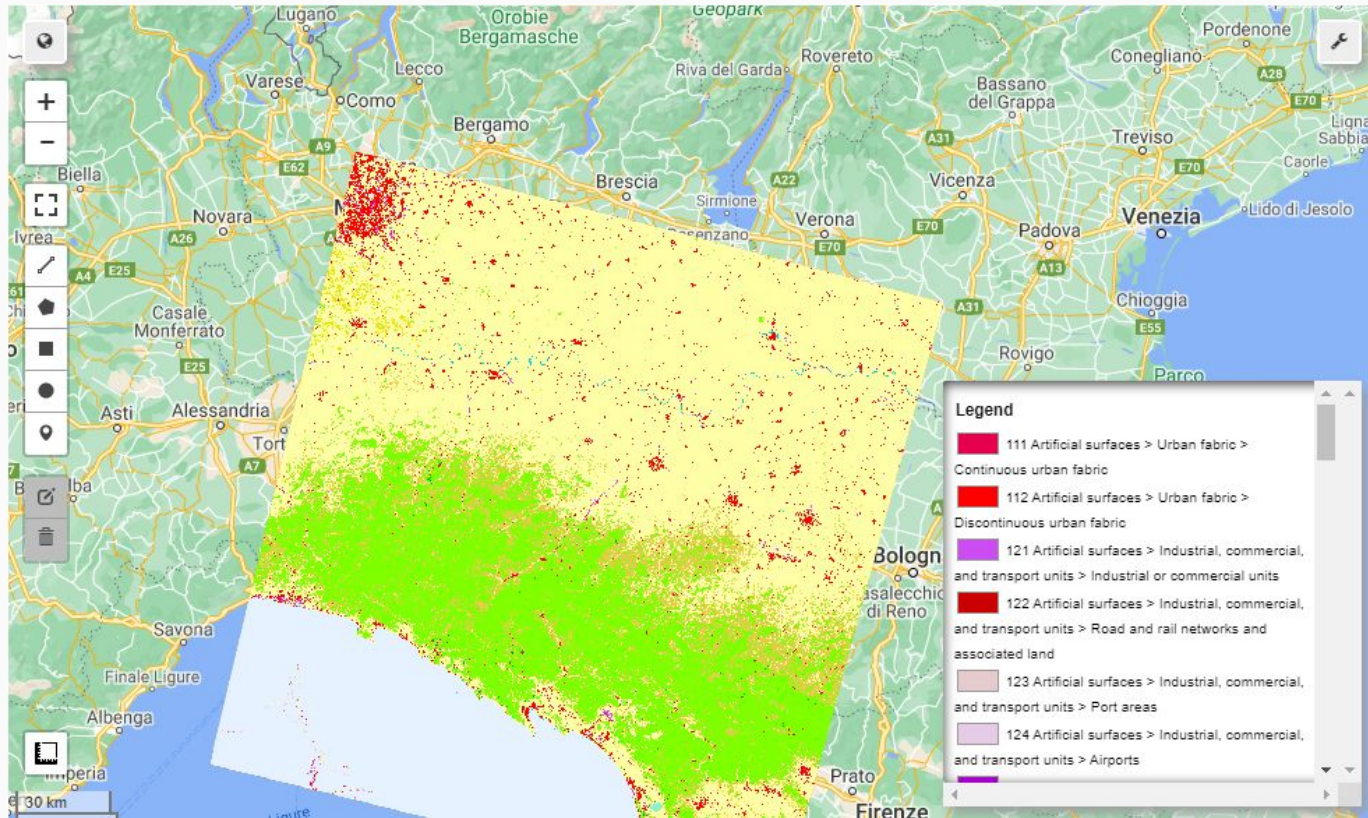
Random Forest visual inspection



Random Forest visual inspection

- By visually inspecting the classification plot we can see that in general the classifier does a pretty good job classifying pixels correctly, however we can notice that most of the misclassifications happen to be in the urban areas
- Probably we can observe this behaviour because the majority of the data points are taken from the Pianura Padana region

Random Forest visual inspection



Random Forest, fewer samples

- By reducing the number of samples a even worse result was produced because having no control on the way points are sampled, doesn't let us compensate for fewer represented classes

Second approach:
Sentinel-2 + Custom classes

Second approach: Sentinel-2 + Custom classes

- Since the first approach didn't work well, I decided to move to the JS Google Earth Engine platform which provides a more friendly interface to interact with the map
- In this case I decided to limit the number of classes to 4 (water, vegetation, soil, urban) so that the sampling procedure was easier and a balanced dataset could be created

Dataset creation

- 100 points for each class were collected by sampling on the map so that the final dataset contained 400 points
- The dataset was then filtered to reduce possible spatial correlation (200m distance)
- The dataset can finally be divided into a training set and into a test (or validation) set

Random Forest: Training and evaluation procedure

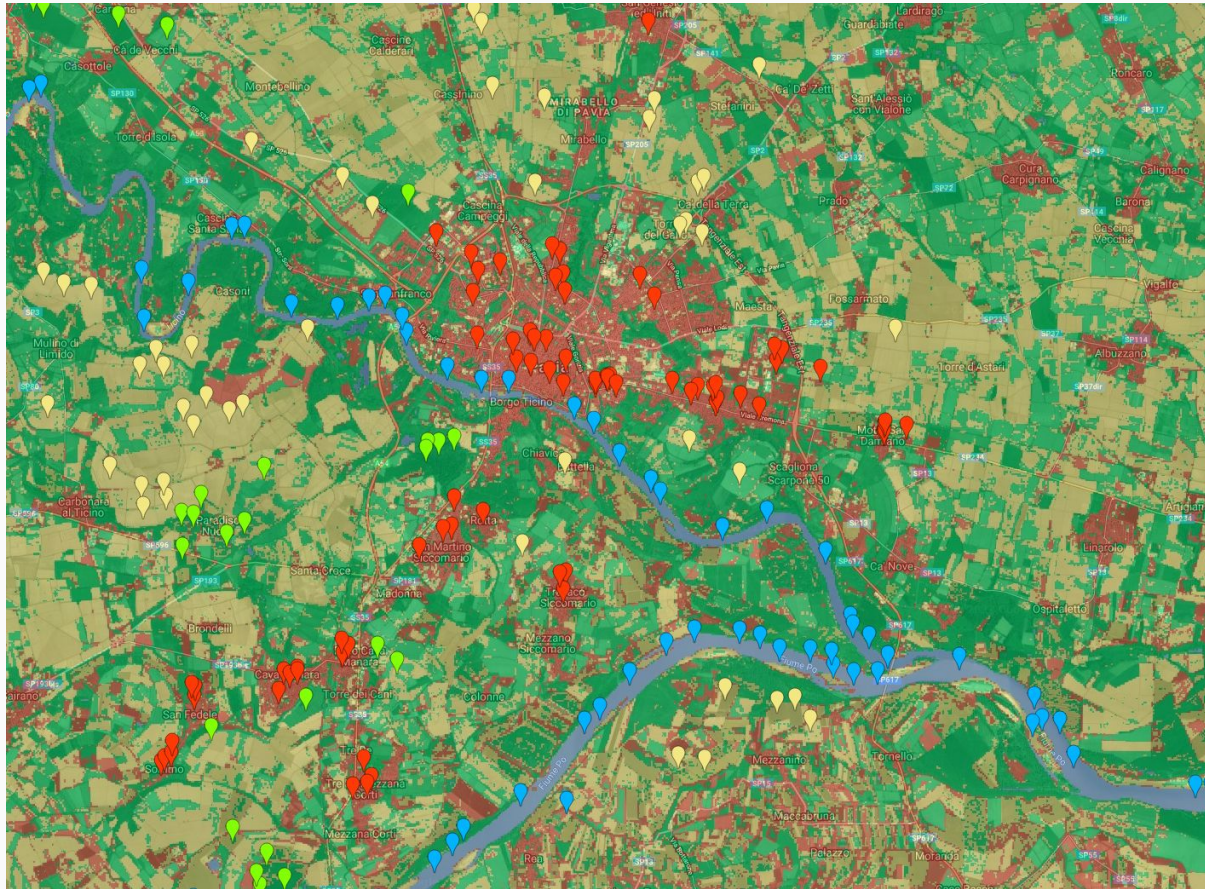
- To assess the model's performance, I set up a grid search method to systematically verify the effect of the parameters' tuning
- For every set of parameters a RF model is trained and evaluated
- Training Accuracy, Test Accuracy and error matrices are recorded for every single model
- The model's parameters we are going to change in the training phase are:
number of decision trees (nTree), number of variables per split (mTry), percentage of dataset that will be dedicated to Tr and Ts

nTree	mTry	Split (Ts)	trAcc	tsAcc
100	3	0.4	1	0.768
100	3	0.3	1	0.785
100	3	0.2	0.995	0.8255
100	3	0.1	0.994	0.829
10	3	0.2	0.97	0.767
50	3	0.2	0.996	0.802
100	3	0.2	1	0.825
500	3	0.2	1	0.825
1000	3	0.2	1	0.825
100	3	0.2	0.98	0.825
100	5	0.2	0.978	0.814
100	6	0.2	1	0.827
100	8	0.2	1	0.823

Random Forest: best parameter combination

- **tsAcc: 0.853**
- trAcc: 0.994
- split: 0.9tr, **0.1ts**
- mTry: **6**
- nTree: **100**

Random Forest final classification



Conclusions

- The model needs to be refined maybe by applying some basic computer vision corrections to get a more precise classification
- More points (sampled in a balanced way) should be used to achieve higher classification performance