Preprocess Guide

December 9, 2021

1 How to use preprocess functions

1.1 `full_feature_matrix.py`

`full_feature_matrix.py` contains function named `create_full_feature_and_target_matrix()` that loops `featureMatrix()`-function and this way creates feature- and target matrixes from given data. After that, the data is going to be pushed into **database** if `pushToDatabase` is set `True` . Else it creates just dataframes and returns them.

1.1.1 Import

[1]:
```
from Preprocess.full_feature_matrix import create_full_feature_and_target_matrix
```

help-function shows you how to use the function and which parameters you can change:

```
help(create_full_feature_and_target_matrix)
```

Default parameters of matrix creation:

```
def create_full_feature_and_target_matrix(fullpathToPairsCsv="/raute_data/Raute/JsonForSchoolPr
                                          pathToPeelJson='/raute_data/Raute/JsonForSchoolProjectTe
                                          pathToDryJson='/raute_data/Raute/JsonForSchoolProjectTes
                                          printProgress=False,
                                          pushToDatabase=True,
                                          host = '172.17.0.2',
                                          user = 'root',
                                          password = 'team1',
                                          database = 'Projekti',
                                          port = 3306,
                                          chunksize=10,
                                          featuresTableName=None,
                                          targetTableName=None,
                                          combinedTableName='PreprocessedData2',
                                          unique = True):
```

1.2 1. Use Case: Feature matrix to dataframe

If you want feature matrix to dataframe, use following code:

[3]:
```
X, Y = create_full_feature_and_target_matrix(fullpathToPairsCsv="/home/jovyan/
↪work/data/nfs_shared_data/Raute/JsonForSchoolProjectTest/VerifiedPairs.csv",
                                             pathToPeelJson='/home/jovyan/work/
↪raute_data/NewPeel/',
                                             pathToDryJson='/home/jovyan/work/
↪raute_data/NewDry/',
                                             printProgress=True,
                                             pushToDatabase=False)
```

```
146/146 dropped nan-rows 2      dropped too big shrinkage rows 7
```

[4]:
```
print(X.shape)
print(Y.shape)
```

```
(137, 87)
(137, 3)
```

1.3 2. Use Case: Feature matrix to database (recommended use)

If you want preprocessed data to straight to the database use following code:

Example:

```
create_full_feature_and_target_matrix(fullpathToPairsCsv = "/raute_data/Raute/JsonForSchoolPro
                                      pathToPeelJson = '/raute_data/Raute/JsonForSchoolProject
                                      pathToDryJson = '/raute_data/Raute/JsonForSchoolProject
                                      printProgress = False,
                                      pushToDatabase = True,
                                      host = '172.17.0.2',
                                      user = 'root',
                                      password = 'team1',
                                      database = 'Projekti',
                                      port = 3306,
                                      chunksize = 10000,
                                      featuresTableName = None,
                                      targetTableName = None,
                                      combinedTableName = 'PreprocessedData',
                                      unique = True)
```

1.3.1 Parameters:

- **fullpathToPairsCsv** = string, DEFAULT = "/home/jovyan/work/data/nfs_shared_data/Raute/JsonForS
- **pathToPeelJson** = string, DEFAULT = '/home/jovyan/work/raute_data/NewPeel/'
- **pathToDryJson** = string, DEFAULT = '/home/jovyan/work/raute_data/NewDry/'
- **printProgress** = bool, if True prints how manieth file is being processed of total
- **pushToDatabase**= bool, if True (DEFAULT) data is pushed to database and nothing is returned, if False full feature and target matrixes are returned
- **chunksize** = int or None, if None data is pushed as a whole to database, otherwise in desired chunksizes

- **featuresTableName**=None or string
- **targetTableName**=None or string, if both featuresTableName and targetTableName has a name, seperate tables will be created for them to database
- **combinedTableName**=string or None (default='PreprocesseddData') if not None, combined table will be created which has both features and target at the same table
- **host** = MariaDB container IP
- **user** = username to your container
- **password** = password to your MariaDB
- **database** = Database name where you want to push the data
- **port** = Port number of your container
- **unique** = If you want to make new tables which dont allow duplicate values then give this True. If unique = True, function will make new tables which wont allow duplicate values. If you have called this once and want to add more data to that specific table, then pass unique = False so it doesn't try to make a new table.

Notes:

- Also if chunksize > amount of data, function pushes all data at once
- Make sure that the paths are defined correctly
- Pushing to database requires direct access to database (SSH-connection IS NOT implemented yet)

2 `feature_matrix.py`

Function `featureMatrix()` creates feature matrix of the **given JSON-file**. If necessary, it's also possible to plot all blocks on the image or just a single block to make sure that the function works properly and view what kind of sheet it is.

This function is dependent by `coordinates.py`, `createColumns.py`, `densitychecker.py`, `readjson.py` and `extractimage.py`.

With `help`-function you can view how to use the function and which parameters you can change:

```
help(featureMatrix)
```

Default parameters:

```
def featureMatrix(dataPath="/home/jovyan/work/raute_data/NewDry/20210505151241_35.json",
                  blockplot=False,
                  datxPath="/home/jovyan/work/data/nfs_shared_data/Raute/ai-2021h2-data/rawdata
                  prints=False):
```

2.0.1 Parameters

- **dataPath :**
  - Path to the JSON-file
  - Example: '/home/jovyan/work/data/nfs_shared_data/Raute/JsonForSchoolProjectTest/Peel/202105
- **blockplot :**
  - Plots all blocks or only one selected block on the sheet's image.

  - Default : False
  - "All" : Plots all nine blocks on image.
  - (1-9) : Plots block of given number.
- **datxPath :**
  - The path to the folder that contains the images that will be extracted for plot
  - Default : /home/jovyan/work/data/nfs_shared_data/Raute/ai-2021h2-data/rawdata/3-Sorvi/koivu/testRun20210505
- **prints :**
  - If True, function prints it's progress e.q. filename. Mainly for debugging.
  - Default : False

2.0.2 Import function

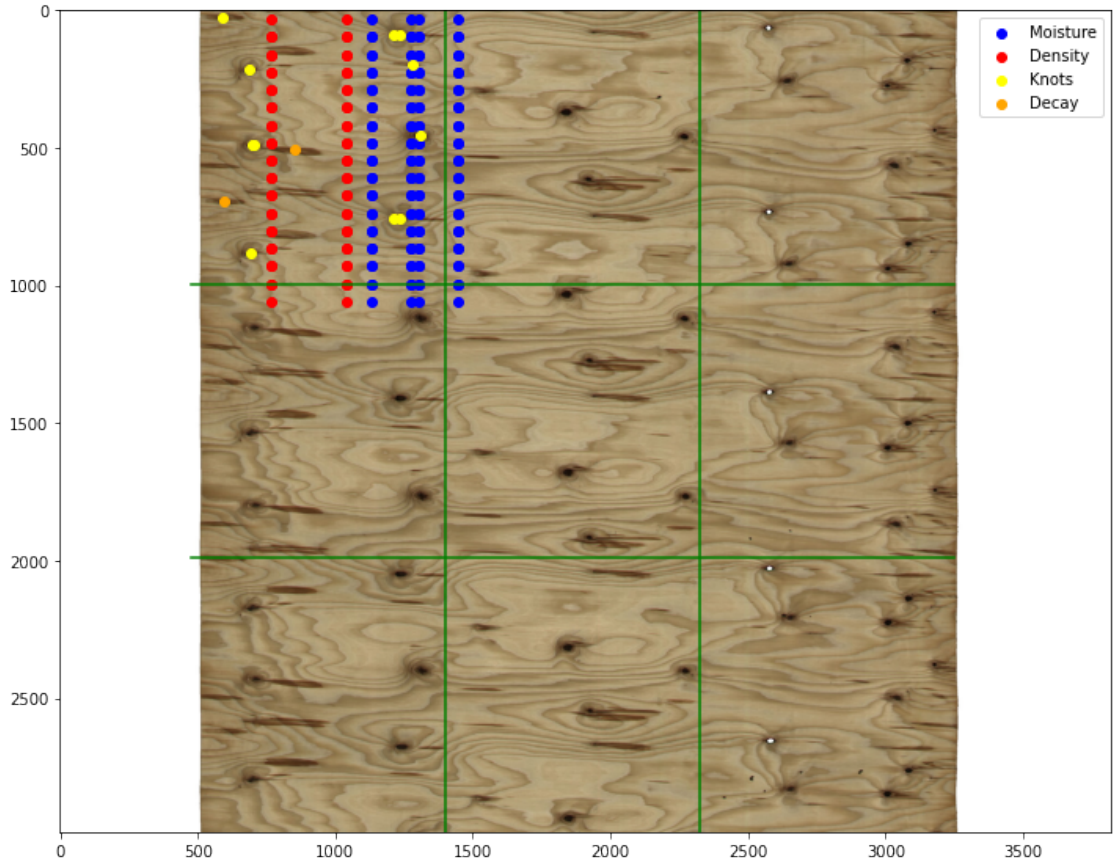
[7]:
```
from Preprocess.feature_matrix import featureMatrix
```

2.0.3 Call function and plot choosen block:

The function can retrieve its own image for that sheet

[7]:
```
features=featureMatrix('/home/jovyan/work/data/nfs_shared_data/Raute/
↪JsonForSchoolProjectTest/Peel/20210505123334_85.json',1)
```

```
Plotting block 1…
```

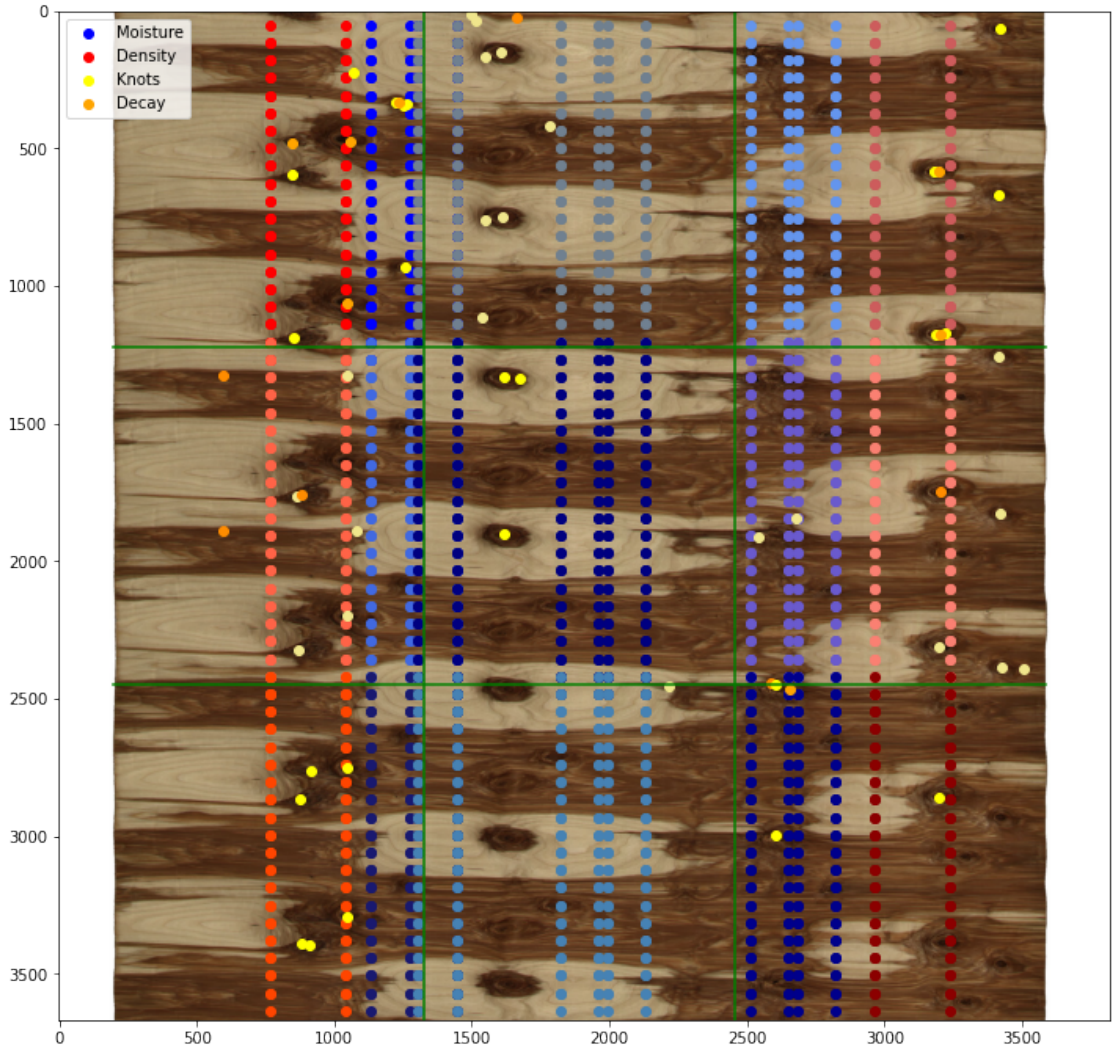


2.0.4 Plot all blocks:

[9]:
```
features = featureMatrix('/home/jovyan/work/raute_data/NewPeel/
↪20210505121149_13.json',"All")
```

```
Plotting All…

Done in 1.76 seconds
```


Moisture
Density
Knots
Decay
0
500
1000
1500
2000
2500
3000
3500