dev_device

September 12, 2024

1 DeviceEngine Class

Dedicated engine for device data, inherited from Core Engine. Each DeviceEngine class object will represent a unique device with its own set of processing parameters and results.

[1]:
```
from src.StreamPort.device.DeviceEngine import DeviceEngine
from src.StreamPort.core.ProjectHeaders import ProjectHeaders
```

[2]:
```
#specify path to get analyses from
base_dir = r'C:\Users\PC0118\Desktop\ExtractedSignals'
```

Creates an empty DeviceEngine object and prints it

[ ]:
```
dev = DeviceEngine(source = base_dir)
dev.print()
```

DeviceEngine object without an explicitly provided source performs all capabilities on files within the current working directory.

[ ]:
```
dev1 = DeviceEngine()
dev1.print()
print(dev1._source)
del dev1
```

2 ProjectHeaders Class

Add project headers. They can be passed as ProjectHeaders objects or dict

[ ]:
```
dev.add_headers(headers = {'name': 'Pressure Curve Analysis', 'author':␣
↪'Sandeep H.'})
dev.print()
```

3 DeviceAnalysis Class

Each DeviceAnalysis object is a child of the Analysis Class. It holds the details of an Analysis for each individual device.

[ ]:
```
from src.StreamPort.device.DeviceAnalysis import DeviceAnalysis

#Creates an empty DeviceAnalysis object and prints it
devAnalysis = DeviceAnalysis()
devAnalysis.print()
```

DeviceEngine's find_analyses() method returns a DeviceAnalysis Object or a list of DeviceAnalysis objects, besides printing the dataframes for each unique Method, paired with the metadata(Date, Runtime) for each curve.

This method makes use of the source variable to accept a path to a directory containing analyses as an argument and find analyses from the target path.

The path can refer to a directory containing data for specific groups of experiments "210812_Gem 2021-08-12 09-49-10" or one such experiment containing its own set of method-related analysis data "210812_Gem–005.D", "210812_Gem–007.D", ..

Read analysis objects from engine.

[ ]:
```
analyses = dev.find_analyses()
```

Each DeviceEngine object has an attribute _method_ids that records all methods encountered in the analysis of the current Device.

[ ]:
```
print(dev._method_ids)
```

And an attribute _history to hold data on all experiments related to this device.

[ ]:
```
dev.print()
```

Add analyses objects that were found using find_analyses() to current device records.

Add analyses in the form of individual DeviceAnalysis objects or a list of such objects.

[10]:
```
dev.add_analyses(analyses)
```

[ ]:
```
dev.print()
```

[ ]:
```
for ana in dev._analyses:
    print("\n")
    print("Analysis Object : \n")
    print(f"Analysis : {ana.print()}")
    print("Data of Analysis : \n")
    print(ana.data)
    print("\n")
```

4 Plot Analyses

DeviceEngine's *plot_analyses()* and *plot_results()* calls each analysis object's respective *plot()* function after dynamically grouping related analyses. Grouping is done on the basis of unique method

id's paired with unique experiment dates. User can set the 'group_by'(str) argument to control how the data is grouped. Defaults to 'method', otherwise 'date'

Plot analyses by calling inbuilt plot function and passing each object's index as argument

Plot analyses by word or subword present in analysis date

[ ]:
```
dev.plot_analyses('Pac', group_by='date')
```

Plot all available analyses by omitting 'analyses' argument Group by defaults to 'method'

[ ]:
```
dev.plot_analyses('Gem', group_by='method')
```

5 ProcessingSettings - Feature Extraction

Create a new ProcessingSettings object

[15]:
```
from src.StreamPort.device.DeviceProcSettings import ExtractPressureFeatures
```

'weighted' argument of ExtractPressureFeatures object can be used to control whether the pressure curves should first be transformed by calculating percentage change between adjacent datapoints. Defaults to False, in which case feature extraction is performed on the raw pressure curves.

[16]:
```
settings = ExtractPressureFeatures(weighted=True)
```

Add processing settings

[ ]:
```
dev.add_settings(settings)
dev.print()
```

Now we run the settings to extract pressure features after adding analyses.

[18]:
```
pressure_features = settings.run(dev)
```

[ ]:
```
print(pressure_features)
```

Add the extracted features to the results (dict) attribute

[20]:
```
dev.add_results(pressure_features)
```

Retrieve the stored results associated with the current object.

6 ProcessingSettings - Seasonal Decomposition

Create a new ProcessingSettings object to extract seasonal components from analyses.

[21]:
```
from src.StreamPort.device.DeviceProcSettings import DecomposeCurves
```

*'period' argument of DecomposeCurves is used to control the window size over which the features are calculated. Defaults to 30 here.

[22]:
```
curve_decompose = DecomposeCurves(period=30)
```

[ ]:
```
dev.add_settings(curve_decompose)
dev.print()
```

[ ]:
```
seasonal_components = curve_decompose.run(dev)
print(seasonal_components)
```

[25]:
```
dev.add_results(seasonal_components)
```

[ ]:
```
dev.get_results(-1)
```

#Each .D folder is an analysis with timestamp

Latest entry in analyses contains most up to date results

7 ProcessingSettings - Fourier Transformation

Create a new ProcessingSettings object to perform Fast Fourier Analysis on raw curve and seasonal component of analyses time decomposition.

[27]:
```
from src.StreamPort.device.DeviceProcSettings import FourierTransform
```

[28]:
```
fourier_transform = FourierTransform()
```

[ ]:
```
dev.add_settings(fourier_transform)
dev.print()
```

[ ]:
```
transformed_seasonal = fourier_transform.run(dev)
print(transformed_seasonal)
```

[31]:
```
dev.add_results(transformed_seasonal)
```

[ ]:
```
dev.get_results(-1)
```

scaled results are unavailable since data has not been scaled yet

Adding features before scaling: scale_features() calls add_extracted_features() before grouping and scaling data.

add_extracted_features() introduces new features that were extracted from the behaviour of the seasonal and noise components of the raw curves in the frequency domain. These frequencies were binned and averaged in different time-windows and added as features.

Additional features added were Idle time of the batch, error in defined vs. measured runtime.

[ ]:
```
dev.print()
```

8 ProcessingSettings - Feature Scaling

Scale extracted and engineered features to improve the quality of the information we get from them. These prove more useful when visually analysing data

[34]:
```
from src.StreamPort.device.DeviceProcSettings import Scaler
```

User selects the type of scaler to be used from preloaded options : 'minmax', 'std'(Standard), 'robust', 'maxabs', 'norm'(Normalizer). Scaler defualts to Normalizer in the absence of an argument.

'replace' argument allows user to replace existing features with scaled features or to create a new entry instead. Defaults to False.

[35]:
```
feature_scaler = Scaler(parameters='std')
```

[ ]:
```
dev.add_settings(feature_scaler)
dev.print()
```

[ ]:
```
scaled_features = feature_scaler.run(dev)
```

[38]:
```
dev.add_results(scaled_features)
```

[ ]:
```
dev.print()
```

9 Plot Results

Plot the computed results of feature extraction for chosen results based on user input to select *base* to extract base features, *decompose* for seasonal decomposition, fourier *transform*

User may also plot the raw pressure curves by omitting the 'features' argument, indicating that the *results* of feature extraction are not to be plotted, just the curves.

[ ]:
```
#this_method = dev._method_ids[6]
this_method = 'Pac'
print(this_method)
```

'group_by' allows user to group data either by 'date' or 'method': 1. 'date' prepares data with weight on experiment date. So matching methods on different dates will not be grouped. 2. 'method' prepares data purely on method and groups all available data for the given method.

[ ]:
```
dev.plot_results(this_method)
```

Select features to plot. Setting 'scaled' argument allows to toggle plots of scaled features or unscaled. Defaults to True.

[ ]:
```
dev.plot_results(results = this_method, features ='base', scaled=True,␣
↪transpose=True, group_by='method', interactive=False)
```

[ ]:
```
dev.plot_results(results = this_method, features ='base', transpose=False,␣
↪interactive=False)
```

use 'interactive' argument to toggle between static and interactive plots

setting type to 'box' enables a box plot of the data. Available options are 'box' and 'scatter' by default

[ ]:
```
dev.plot_results(results = this_method, features ='transform')
```

10 MachineLearning - Isolation Forest for preliminary classification

ADD CLASS LABELS TO ANALYSIS OBJECTS AFTER FEATURE ANALYSIS. FIRST ANALYSIS '001-blank' is assigned a separate class of ML operations due to it being a systematic fault.

classify() dynamically assigns class labels through MLEngine's make_iso_forest() to all analyses encountered and classified

First, create a MachineLearningEngine object to enable ML ops on prepared data.

[45]:
```
from src.StreamPort.ml.MachineLearningEngine import MachineLearningEngine
from src.StreamPort.ml.MachineLearningAnalysis import MachineLearningAnalysis
from src.StreamPort.ml.MachineLearningProcessingSettings import␣
 ↪MakeModelIsoForest
from src.StreamPort.ml.MachineLearningProcessingSettings import MakeModelPCASKL
```

[46]:
```
ml_engine = MachineLearningEngine()
```

random_state(int) argument can be specified to reproduce results. Defaults to None, sets a random seed.

[63]:
```
iso_forest = MakeModelIsoForest(dev, random_state=22)
```

[ ]:
```
ml_engine.add_settings(iso_forest)
ml_engine.print()
```

[ ]:
```
method_objects = iso_forest.run(ml_engine)
```

11 MachineLearning - PCA

make_iso_forest() of MLEngine class automatically creates sub-objects of MLEngine class for each encountered group of analyses per unique method after performing iso_forest and plotting results. Can be modified to save results later

[53]:
```
pca = MakeModelPCASKL(n_components = 2, center_data= True)
```

[ ]:
```
import webbrowser
for obj in method_objects:
    obj.add_settings(pca)
    obj.print()
    pca_scores = pca.run(obj)
    obj.add_results(pca_scores)
    obj.plot_pca()
```