Hello Radiomics example: using the feature extractor to calculate features

Install the PyRadiomics package read more about it here (https://pyradiomics.readthedocs.io/en/latest/ (https://pyradiomics.readthedocs.io/en/latest/))

In [ ]:
```
#!pip install pyradiomics
```

This example shows how to use the radiomics package and the feature extractor. The feature extractor handles preprocessing, and then calls the needed featureclasses to calculate the features. It is also possible to directly instantiate the feature classes. However, this is not recommended for use outside debugging or development. For more information, see `helloFeatureClass` .

In [ ]:
```
from __future__ import print_function
import sys
import os
import logging
import six
from radiomics import featureextractor, getFeatureClasses
import radiomics
```

Setting up logging

Regulate verbosity of PyRadiomics (outputs to stderr)

In [ ]:
```
# Regulate verbosity with radiomics.setVerbosity
# radiomics.setVerbosity(logging.INFO)  # Use logging.DEBUG for maxi
mum output, default verbosity level = WARNING
```

Set up logging to a log file

In [ ]:
```
# Get the PyRadiomics logger (default log-level = INFO)
logger = radiomics.logger
logger.setLevel(logging.DEBUG)  # set level to DEBUG to include debu
g log messages in log file

# Write out all log entries to a file
handler = logging.FileHandler(filename='testLog.txt', mode='w')
formatter = logging.Formatter('%(levelname)s:%(name)s: %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
```

Getting the testcase

Test cases can be downloaded to temporary files. This is handled by the `radiomics.getTestCase()` function, which checks if the requested test case is available and if not, downloads it. It returns a tuple with the location of the image and mask of the requested test case, or (None, None) if it fails.

Alternatively, if the data is available somewhere locally, this directory can be passed as a second argument to `radiomics.getTestCase()`. If that directory does not exist or does not contain the testcase, functionality reverts to default and tries to download the test data.

If getting the test case fails, PyRadiomics will log an error explaining the cause.

In [ ]:
```
featureClasses = getFeatureClasses()
imageName, maskName = radiomics.getTestCase('brain1')

if imageName is None or maskName is None:  # Something went wrong, i
n this case PyRadiomics will also log an error
    raise Exception('Error getting testcase!')  # Raise exception to
prevent cells below from running in case of "run all"
```

Initializing the feature extractor

Extraction Settings

In [ ]:
```
# Use a parameter file, this customizes the extraction settings and
also specifies the input image types to use and which features shoul
d be extracted
params = os.path.join('..', 'examples', 'exampleSettings', 'Params.y
aml')

extractor = featureextractor.RadiomicsFeatureExtractor(params)
```

In [ ]:
```
# Alternative: use hardcoded settings (separate for settings, input
image types and enabled features)
settings = {}
settings['binWidth'] = 25
settings['resampledPixelSpacing'] = None
# settings['resampledPixelSpacing'] = [3, 3, 3]  # This is an exampl
e for defining resampling (voxels with size 3x3x3mm)
settings['interpolator'] = 'sitkBSpline'
settings['verbose'] = True

extractor = featureextractor.RadiomicsFeatureExtractor(**settings)
```

Input images: applying filters

In [ ]:
```
# By default, only 'Original' (no filter applied) is enabled. Option
ally enable some image types:

# extractor.enableImageTypeByName('Wavelet')
# extractor.enableImageTypeByName('LoG', customArgs={'sigma':[3.0]})
# extractor.enableImageTypeByName('Square')
# extractor.enableImageTypeByName('SquareRoot')
# extractor.enableImageTypeByName('Exponential')
# extractor.enableImageTypeByName('Logarithm')

# Alternative; set filters in one operation
# This updates current enabled image types, i.e. overwrites custom s
ettings specified per filter.
# However, image types already enabled, but not passed in this call,
are not disabled or altered.

# extractor.enableImageTypes(Wavelet={}, LoG={'sigma':[3.0]})

print('Enabled input images:')
for imageType in extractor.enabledImagetypes.keys():
    print('\t' + imageType)
```

Feature classes: setting which feature(classes) need to be calculated

In [ ]:
```
# Disable all classes
extractor.disableAllFeatures()

# Enable all features in firstorder
extractor.enableFeatureClassByName('firstorder')

# Alternative; only enable 'Mean' and 'Skewness' features in firstor
der
# extractor.enableFeaturesByName(firstorder=['Mean', 'Skewness'])
```

Getting the docstrings of the active features

In [ ]:
```
print('Active features:')
for cls, features in six.iteritems(extractor.enabledFeatures):
    if len(features) == 0:
        features = [f for f, deprecated in six.iteritems(featureClas
ses[cls].getFeatureNames()) if not deprecated]
    for f in features:
        print(f)
        print(getattr(featureClasses[cls], 'get%sFeatureValue' % f).
__doc__)
```

Calculating the values of the active features

In [ ]:
```
print('Calculating features')
featureVector = extractor.execute(imageName, maskName)
```

In [ ]:
```
# Show output
for featureName in featureVector.keys():
    print('Computed %s: %s' % (featureName, featureVector[featureNam
e]))
```