gc-ims-tools v. 0.1.6 Cheat Sheet

CHARISMA

https://www.charisma.hs-mannheim.de/charisma.html

Documentation

This cheat sheet summarizes functionality and API. For detailed documentation about individual methods please refer to: `https://charisma-mannheim.github.io/gc-ims-tools/build/html/index.html` or the docstrings.

Installation and Usage

gc-ims-tools can be installed with pip: `pip install gc-ims-tools`
Access content by importing ims namespace: `import ims`

Data Structures

Spectrum Class

`ims.Spectrum`
Represents a single GC-IMS spectrum and contains the following attributes:

`name`	File name, unique identifier.
`values`	Recorded intensity matrix.
`ret_time`	GC retention time.
`drift_time`	IMS drift time.
`time`	Timestamp when the spectrum was recorded.
`peak_table`	Peak table calculated by find_peaks method.

Methods that can be applied per spectrum are defined here. Use one of the read_... methods from the I/O section as a constructor.

Dataset Class

`ims.Dataset`
Coordinates many GC-IMS spectra (instances of `ims.Spectrum` class) with labels and sample names. Contains the following attributes:

`data`	List of GC-IMS spectra.
`name`	Name for the dataset, identifier.
`files`	List with file names.
`samples`	List with sample names.
`labels`	List with labels, can be categorical or numerical.

Methods that require multiple spectra are defined here. `ims.Spectrum` methods are applied to all spectra in the data set. Use one of the read_... methods from the I/O section as a constructor.

File I/O

.mea Files

.mea is the file format G.A.S Dortmund instruments use.

`ims.Spectrum.read_mea`	Reads a single .mea file.
`ims.Dataset.read_mea`	Reads every .mea file in a folder.

Example

```
>>> sample = ims.Spectrum.read_mea("sample.mea")
>>> print(sample)
GC-IMS Spectrum: sample

>>> ds = ims.Dataset.read_mea("IMS_data")
>>> print(ds)
Dataset: IMS_data, 58 Spectra
```

.hdf5 Files

The .hdf5 methods are useful to save and read preprocessed data.

`ims.Spectrum.to_hdf5`	Saves a single spectrum as .hdf5 file.
`ims.Spectrum.read_hdf5`	Reads .hdf5 file from method above.
`ims.Dataset.to_hdf5`	Saves the dataset as .hdf5 file.
`ims.Dataset.read_hdf5`	Reads .hdf5 file from method above.

Example

```
>>> ds.to_hdf5() # uses name attribute as file name
>>> ds = ims.Dataset.read_hdf5("IMS_Data.hdf5")
```

.csv Files

.csv file readers are included as a general backup for non G.A.S instruments. Binary file formats are preferred due to much higher read and write speeds.

`ims.Spectrum.read_csv`	Reads generic .csv file.
`ims.Dataset.read_csv`	Reads every .csv in folder.

Indexing and Slicing

The `ims.Dataset` class can be indexed and sliced with square brackets similar to Python lists. Indices can be an integer, or collections of integers. To select, add or drop spectra by label or sample name use the following methods:

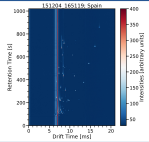
`ims.Dataset.select`	Selects subset with sample or label as key.
`ims.Dataset.drop`	Drops subset with sample or label as key.
`ims.Dataset.add`	Adds spectrum to dataset.
`ims.Dataset.groupby`	Groups spectra by either label or sample and returns a Dataset instance for each group.

Example

```
>>> # returns spectrum at index 0
>>> spectrum = ds[0]

>>> # returns new Dataset with selected spectra
>>> indices = [3, 5, 8, 13]
>>> new_ds = ds[indices]

>>> # returns new Dataset with all samples from Group A
>>> new_ds = ds.select(label="Group A")

>>> # drops all spectra belonging to Sample A
>>> new_ds = ds.drop(sample="Sample A")
```

Visualization

Visualizations are built using *matplotlib* and are fully customizable and compatible with other libraries like *seaborn*.

`ims.Spectrum.plot`	
`ims.Dataset.plot`	Plots selected spectrum, adds label to title.
`ims.Spectrum.export_plot`	Saves plot as image file.
`ims.Dataset.export_plots`	Saves plot as image file for all spectra.

Example

```
>>> sample.plot()
>>> ds.plot(5) # plots the spectrum with index 5

>>> # Scale text font, marker size etc.
>>> import seaborn as sns
>>> sns.set_context("talk")

>>> # export latest figure with custom specifications
>>> import matplotlib.pyplot as plt
>>> plt.savefig("sample.jpg", dpi=300)
```

Preprocessing

All preprocessing methods work in-place.

Spectrum

`ims.Spectrum.cut_dt`	Cuts spectrum in drift time dimension with specified start and stop values.
`ims.Spectrum.cut_rt`	Cuts spectrum in retention time dimension with specified start and stop values.
`ims.Spectrum.binning`	Applies binning by calculating means of every n values in both dimension.
`ims.Spectrum.tophat`	Applies a tophat filter for baseline correction.
`ims.Spectrum.sub_first_row`	Subtracts the first row from all others (baseline correction).
`ims.Spectrum.asymcorr`	Asymmetric least squares baseline correction.
`ims.Spectrum.savgol`	Savitzky-Golay smoothing.

Example

```
>>> sample.binning(2).cut_dt(5, 15).cut_rt(100, 900)
>>> sample.plot() # plot spectrum to see the changes
```

Dataset

All preprocessing methods from the Spectrum class are also available in the Dataset class and are applied to all spectra.

`ims.Dataset.interp_riprel`	Interpolates all spectra to a new RIP-relative drift time axis to correct shifts in the drift time (alignment).
`ims.Dataset.mean`	Calculates means for repeat determinations.
`ims.Dataset.scaling`	Applies different scaling methods e.g. *pareto* and mean centering.

Example

```
>>> # cutting just after the RIP on relative axis
>>> ds.interp_riprel().mean().cut_dt(1.05, 2)
```

Peak detection

The Spectrum class features automated peak detection based on persistent homology.

`ims.Spectrum.find_peaks`	Detects peaks and adds the peak table attribute.
`ims.Spectrum.plot_peaks`	Plots chromatogram with peak labels.
`ims.Spectrum.plot_persistence`	Scatter plot with birth and death levels for each peak.
`ims.Spectrum.watershed_segmentation`	Threshold dependent watershed segmentation for each detected peak.

Example

```
>>> # Call find_peaks method and print peak table
>>> sample.find_peaks()
>>> print(sample.peak_table)
>>>
>>> # plot GC-IMS data with peak labels
>>> sample.plot_peaks()
```

Validation

The Dataset class can return formatted features and labels for different validation strategies.

`ims.Dataset.get_xy`	Returns formatted features and labels.
`ims.Dataset.train_test_split`	Splits data into training and test sets and returns features and labels for each.
`ims.Dataset.kfold_split`	Returns iterator for kfold cross-validation.
`ims.Dataset.shuffle_split`	Returns iterator for monte-carlo cross-validation.
`ims.Dataset.bootstrap`	Returns iterator for sampling with replacement.
`ims.Dataset.leave_one_out`	Returns iterator for leave one out validation.

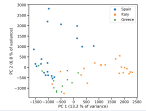
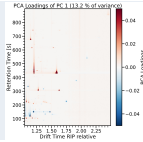
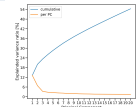
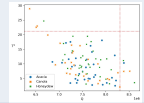
Example

```
>>> # get unfolded feature matrix X and labels y
>>> X, y = ds.get_xy(flatten=True)

>>> # validation loop with any model and error metric
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.metrics import accuracy_score
>>> model = RandomForestClassifier()
>>> accuracy = []
>>> for X_train, X_test, y_train, y_test in ds.\
kfold_split():
>>>     model.fit(X_train, y_train)
>>>     y_pred = model.predict(X_test)
>>>     accuracy.append(accuracy_score(y_test, y_pred))
```

Scripted Workflows

The included statistical work-flows are based on scikit-learn algorithms and add useful visualizations and variable selection tools. They share a common API. An instance of the `ims.Dataset` class must be provided during instantiation to prebuild the plots. The data is then fitted using the fit method. The advantage is that different resampling strategies are more transparent than if the model would use the data from the data set directly and do the validation internally.

PCA

`ims.PCA_Model.plot`	
`ims.PCA_Model.plot_loadings`	
`ims.PCA_Model.scree_plot`	
`ims.PCA_Model.Tsq_Q_plot`	

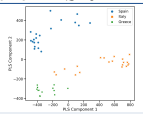
Example

```
>>> pca = ims.PCA_Model(ds, n_components=20)
>>> pca.fit(X) # from get_xy method
>>> # scatter plot with first two principal components
>>> pca.plot()
>>> # scatter plot af component 3 and 4
>>> pca.plot(3, 4)
>>> # plots loadings of second component
>>> pca.plot_loadings(2)
>>> # plots Tsq values and Q residuals
>>> pca.Tsq_Q_plot()
```

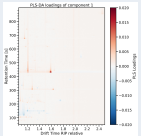
PLS

Both PLS regression and classification with PLS-DA are implemented. The PLS-DA methods are also available for the PLSR.

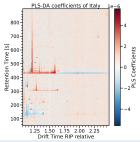
`ims.PLS_DA.plot`



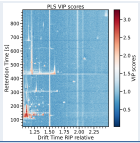
`ims.PLS_DA.plot_loadings`



`ims.PLS_DA.plot_coefficients`



`ims.PLS_DA.plot_vip_scores`



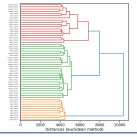
Example

```
>>> plsda = ims.PLS_DA(ds, n_components=5)
>>> # from train_test_split method
>>> plsda.fit(X_train, y_train)
>>> y_pred = plsda.predict(X_test)
>>> plsda.plot() # scatter plot
>>> plsda.plot_vip_scores()
```

HCA

Hierarchical cluster analysis.

`ims.HCA.plot_dendrogram`



Example

```
>>> hca = ims.HCA(ds, linkage="ward")
>>> hca.fit(X)
>>> hca.plot_dendrogram()
```