pyChemometrics

Release 0.13.5

Nat

Jan 14, 2021

Contents

Contents:

CHAPTER 1

Using the pyChemometrics objects

pyChemometrics is a python 3.5 library for multivariate chemometric data analysis.

The main objects ChemometricsPCA, ChemometricsPLS and ChemometricsPLSDA consist of wrappers for scikit-learn Principal Component Analysis and Partial Least Squares Regression objects. They have been made to mimic as much as possible scikit-learn classifiers, from their internal properties, and therefore can be interfaced with other components of scikit-learn, such as the sklearn::*Pipeline*.

These wrappers contain implementations of various routines and metrics commonly seen in the Chemometric and metabonomic literature. PRESS and Q2Y estimation, permutation testing, Hotelling T2 for outlier detection of scores, VIP scores for variable importance. Pareto and Unit-Variance scaling.

Each of these objects uses ChemometricsScaler objects to automatically handle the scaling of the X and Y data matrices.

1.1 Scaling

The `ChemometricsScaler` object handles the scaling of the data matrices. The main s The data is always. The choice of the power determines the type of scaling. For example, scaling_power = 0 performs column centering only, scaling_power = 1/2 Pareto scaling and scaling_power = 1 UV (Unit Variance scaling or standardisation).

`ChemometricsPCA` object. The scaler parameter expects a `ChemometricsScaler` with the default options and Unit-Variance scaling

```
pca_model = pyChemometrics.ChemometricsPCA(...)
```

The pyChemometrics objects contain methods similar to the ones defined in the scikit-learn Transformer, Classifier and Regressor Mixins, for example, .fit, .transform , .predict and .score.

```
pca_model.fit(X) # Obtain the scores (T), the lower dimensional representation of data.
t_scores = pca_model.transform(X) # Obtain the reconstructed dataset from the T scores.
pca_model.inverse_transform(scores)
```

1.2 Principal Component Analysis

Principal Component Analysis is provided by the *ChemometricsPCA* object.

ChemometricsPCA object. The scaler parameter expects a *ChemometricsScaler* with the default options and Unit-Variance scaling

pca_model = pyChemometrics.ChemometricsPCA(. . .)

pca_model.fit(X)

t_scores pca_model.transform(X)

pca_model.inverse_transform(scores)

The scores and loadings obtained for each component upon calling the .fit method are set as atributes of the model.

The modelParameters dictionary contains the following keys:

- VarExp: Total variance explained by the model, per component
- VarExpRatio: % of variance explained, per component
- R2X: The variance explained by the model in the fitting/training set. Calculated using the model residuals.
- S0: The denominator for calculation of the Normalised DmodX score.

Performing model cross_validation using the *cross_validation()* method generates another dictionary atribute, cvParameters. These contain the mean and standard deviation values obtained from the multiple folds or sampling repeats performed the cross-validation, and if cross_validation method was called with outputdist = True, also the whole distribution obtained by CV for each parameter.

The cvParameters dictionary contains these keys:

- Mean_Loadings: Average loading vectors during cross-validation
- Stdev_Loadings: Standard deviation of the loading vectors

If the outputdist option is set to True when performing cross validation, cvParameters will contain extra keys with numpy.ndarrays containing all the model parameters (scores, loadings, goodness of fit metrics, etc) obtained for each model fitted during CV.

The main The methods provided by these objects The pyChemometrics objects follow a similar logic Similarly to scikit-learn:

1.3 Partial Least Squares Regression

The standard Partial Least Squares object

The scores and loadings obtained for each component upon calling the .fit method are set as atributes of the model.

- scores_t:
- scores_u:
- weights_w:
- weights_c:
- loadings_p:
- loadings_q:
- rotations_ws:

- rotations_cs:
- b_u:
- b_t:
- beta_coeffs:
- logistic_coefs:
- n_classes:

The modelParameters dictionary contains the following keys:

- R2Y: Total variance explained by the model, per component
- R2X: % of variance explained, per component
- SSX:
- SSY:
- SSXcomp: The variance explained by the model in the fitting/training set. Calculated using the model residuals.
- SSYcomp: The denominator for calculation of the Normalised DmodX score.

Performing model cross_validation using the *cross_validation()* method generates another dictionary atribute, cvParameters. These contain the mean and standard deviation values obtained from the multiple folds or sampling repeats performed the cross-validation, and if cross_validation method was called with outputdist = True, also the whole distribution obtained by CV for each parameter.

The cvParameters dictionary contains these keys:

- Mean_Loadings: Average loading vectors during cross-validation
- Stdev_Loadings: Standard deviation of the loading vectors

If the outputdist option is set to True when performing cross validation, cvParameters will contain extra keys with numpy.ndarrays containing all the model parameters (scores, loadings, goodness of fit metrics, etc) obtained for each model fitted during CV.

`ChemometricsPLS`

1.4 Partial Least Squares - Discriminant Analysis

The *ChemometricsPLSDA* object shares many features with the `ChemometricsPLS` object.

Calling the fit method will fill in these

- scores_t:
- scores_u:
- weights_w:
- weights_c:
- loadings_p:
- loadings_q:
- rotations_ws:
- rotations_cs:

- b_u:
- b_t:
- beta_coeffs:
- logistic_coefs:
- n_classes:

However, this object expects either a singly Y vector containing, or a dummy matrix. The singly Y vector encoding class membership is re-coded as a dummy matrix of dimensions [n observations x m classes] as part of the algorithm.

The scores and loadings obtained for each component upon calling the .fit method are set as atributes of the model.

The modelParameters dictionary attributes are contains the following keys: The 'PLS' subdictionary contains all the values pertaining to the PLS regression algorithm. - R2Y: Total variance explained by the model, per component - R2X: % of variance explained, per component - SSX: - SSY: - SSXcomp: The variance explained by the model in the fitting/training set. Calculated using the model residuals. - SSYcomp: The denominator for calculation of the Normalised DmodX score. The 'DA' subdictionary contains the classification metrics obtained by scoring the class predictions with the known truth. - Balanced accuracy: - F1 measure: - Precision: - Recall: - ROC curve: - AUC: - 01-Loss: - MCC:

Performing model cross_validation using the *cross_validation()* method generates another dictionary atribute, cvParameters. These contain the mean and standard deviation values obtained from the multiple folds or sampling repeats performed the cross-validation, and if cross_validation method was called with outputdist = True, also the whole distribution obtained by CV for each parameter.

The cvParameters dictionary contains these keys:

- Mean_Loadings: Average loading vectors during cross-validation
- Stdev_Loadings: Standard deviation of the loading vectors

Additionaly, the discriminant analysis also contains the mean and standard deviation parameters for the DA component.

- Mean_Accuracy:
- Stdev_Accuracy:

If the outputdist option is set to True when performing cross validation, cvParameters will contain extra keys with numpy.ndarrays containing all the model parameters (scores, loadings, goodness of fit metrics, etc) obtained for each model fitted during CV.

1.5 Partial Least Squares - Logistic Regression

The *ChemometricsPLS_Logistic* object shares many features with the `ChemometricsPLS` object.

- scores_t:
- scores_u:
- weights_w:
- weights_c:
- loadings_p:
- loadings_q:
- rotations_ws:
- rotations_cs:

- b_u:
- b_t:
- beta_coeffs:
- logistic_coefs:
- n_classes:

Calling the fit method will fill in these

1.6 Partial Least Squares - Linear Discriminant Analysis

The `ChemometricsPLS_LDA` object shares many features with the `ChemometricsPLS_LDA` object.

- scores_t:
- scores_u:
- weights_w:
- weights_c:
- loadings_p:
- loadings_q:
- rotations_ws:
- rotations_cs:
- b_u:
- b_t:
- beta_coeffs:
- logistic_coefs:
- n_classes:

Calling the fit method will fill in these

CHAPTER 2

pyChemometrics objects

Reference guide for the pyChemometrics objects.

2.1 ChemometricsPCA

class pyChemometrics.**ChemometricsPCA**(*ncomps=2*, *pca_algorithm=<class 'sklearn.decomposition._pca.PCA'>*, *scaler=ChemometricsScaler()*, ***pca_type_kwargs*)

ChemometricsPCA object - Wrapper for sklearn.decomposition PCA algorithms, with tailored methods for Chemometric Data analysis.

Parameters

- **ncomps** (*int*) – Number of PCA components desired.
- **pca_algorithm** (*sklearn.decomposition._BasePCA*) – scikit-learn PCA models (inheriting from _BasePCA).
- **scaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None*) – The object which will handle data scaling.
- **pca_type_kwargs** (*kwargs*) – Keyword arguments to be passed during initialization of pca_algorithm.

Raises **TypeError** – If the pca_algorithm or scaler objects are not of the right class.

fit(*x*, ***fit_params*)

Perform model fitting on the provided x data matrix and calculate basic goodness-of-fit metrics. Equivalent to scikit-learn's default BaseEstimator method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PCA model.

- **fit_params** (*kwargs*) – Keyword arguments to be passed to the .fit() method of the core sklearn model.

Raises **ValueError** – If any problem occurs during fitting.

fit_transform(*x*, ***fit_params*)

Fit a model and return the scores, as per the scikit-learn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit and project.
- **fit_params** (*kwargs*) – Optional keyword arguments to be passed to the fit method.

Returns PCA projections (scores) corresponding to the samples in X.

Return type numpy.ndarray, shape [n_samples, n_comps]

Raises **ValueError** – If there are problems with the input or during model fitting.

transform(*x*)

Calculate the projections (scores) of the x data matrix. Similar to scikit-learn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit and project.
- **transform_params** (*kwargs*) – Optional keyword arguments to be passed to the transform method.

Returns PCA projections (scores) corresponding to the samples in X.

Return type numpy.ndarray, shape [n_samples, n_comps]

Raises **ValueError** – If there are problems with the input or during model fitting.

score(*x*, *sample_weight=None*)

Return the average log-likelihood of all samples. Same as the underlying score method from the scikit-learn PCA objects.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to score model on.
- **sample_weight** (*numpy.ndarray*) – Optional sample weights during scoring.

Returns Average log-likelihood over all samples.

Return type float

Raises **ValueError** – if the data matrix x provided is invalid.

inverse_transform(*scores*)

Transform scores to the original data space using the principal component loadings. Similar to scikit-learn's default TransformerMixin method.

Parameters **scores** (*numpy.ndarray, shape [n_samples, n_comps]*) – The projections (scores) to be converted back to the original data space.

Returns Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features]

Raises **ValueError** – If the dimensions of score mismatch the number of components in the model.

hotelling_T2(*comps=None*, *alpha=0.05*)

Obtain the parameters for the Hotelling T2 ellipse at the desired significance level.

Parameters

- **comps** (*list*) –
- **alpha** (*float*) – Significance level

Returns The Hotelling T2 ellipsoid radii at vertex

Return type numpy.ndarray

Raises

- **AtributeError** – If the model is not fitted
- **ValueError** – If the components requested are higher than the number of components in the model
- **TypeError** – If comps is not None or list/numpy 1d array and alpha a float

x_residuals(*x*, *scale=True*)

Parameters

- **x** – data matrix [n samples, m variables]
- **scale** – Return the residuals in the scale the model is using or in the raw data scale

Returns X matrix model residuals

dmodx(*x*)

Normalised DmodX measure

Parameters **x** – data matrix [n samples, m variables]

Returns The Normalised DmodX measure for each sample

leverages()

Calculate the leverages for each observation

Returns The leverage (H) for each observation

Return type numpy.ndarray

cross_validation(*x*, *cv_method=KFold(n_splits=7, random_state=None, shuffle=True)*, *outputdist=False*, *press_impute=True*)

Cross-validation method for the model. Calculates cross-validated estimates for Q2X and other model parameters using row-wise cross validation.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix.
- **cv_method** (*BaseCrossValidator*) – An instance of a scikit-learn CrossValidator object.
- **outputdist** (*bool*) – Output the whole distribution for the cross validated parameters.

Useful when using ShuffleSplit or CrossValidators other than KFold. :param bool press_impute: Use imputation of test set observations instead of row wise cross-validation. Slower but more reliable. :return: Adds a dictionary cvParameters to the object, containing the cross validation results :rtype: dict :raise TypeError: If the cv_method passed is not a scikit-learn CrossValidator object. :raise ValueError: If the x data matrix is invalid.

outlier(*x*, *comps=None*, *measure='T2'*, *alpha=0.05*)
Use the Hotelling T2 or DmodX measure and F statistic to screen for outlier candidates.

Parameters

- **x** – Data matrix [n samples, m variables]
- **comps** – Which components to use (for Hotelling T2 only)
- **measure** – Hotelling T2 or DmodX
- **alpha** – Significance level

Returns List with row indices of X matrix

permutationtest_loadings(*x*, *nperms=1000*)
Permutation test to assess significance of magnitude of value for variable in component loading vector. Can be used to test importance of variable to the loading vector.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix.
- **nperms** (*int*) – Number of permutations.

Returns Permuted null distribution for loading vector values.

Return type numpy.ndarray, shape [ncomps, n_perms, n_features]

Raises **ValueError** – If there is a problem with the input x data or during the procedure.

permutationtest_components(*x*, *nperms=1000*)
Unfinished Permutation test for a whole component. Also outputs permuted null distributions for the loadings.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix.
- **nperms** (*int*) – Number of permutations.

Returns Permuted null distribution for the component metrics (VarianceExplained and R2).

Return type numpy.ndarray, shape [ncomps, n_perms, n_features]

Raises **ValueError** – If there is a problem with the input data.

2.2 ChemometricsPLS

class pyChemometrics.**ChemometricsPLS**(*ncomps=2*, *pls_algorithm=<class 'sklearn.cross_decomposition._pls.PLSRegression'>*, *xscaler=ChemometricsScaler()*, *yscaler=None*, ***pls_type_kwargs*)
ChemometricsPLS object - Wrapper for sklearn.cross_decomposition PLS algorithms, with tailored methods for Chemometric Data analysis.

Parameters

- **ncomps** (*int*) – Number of PLS components desired.
- **pls_algorithm** (*sklearn._PLS*) – Scikit-learn PLS algorithm to use - PLSRegression or PLSCanonical are supported.

- **xscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for X data matrix.
- **yscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for the Y data vector/matrix.
- **pls_type_kwargs** (*kwargs*) – Keyword arguments to be passed during initialization of pls_algorithm.

Raises **TypeError** – If the pca_algorithm or scaler objects are not of the right class.

fit(*x*, *y*, ***fit_params*)

Perform model fitting on the provided x and y data and calculate basic goodness-of-fit metrics. Similar to scikit-learn's BaseEstimator method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Keyword arguments to be passed to the .fit() method of the core sklearn model.

Raises **ValueError** – If any problem occurs during fitting.

fit_transform(*x*, *y*, ***fit_params*)

Fit a model to supplied data and return the scores. Equivalent to scikit-learn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Optional keyword arguments to be passed to the pls_algorithm .fit() method.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple of numpy.ndarray, shape [[n_tscores], [n_uscores]]

Raises **ValueError** – If any problem occurs during fitting.

transform(*x=None*, *y=None*)

Calculate the scores for a data block from the original data. Equivalent to sklearn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple with 2 numpy.ndarray, shape [n_samples, n_comps]

Raises

- **ValueError** – If dimensions of input data are mismatched.
- **AttributeError** – When calling the method before the model is fitted.

inverse_transform(*t=None*, *u=None*)

Transform scores to the original data space using their corresponding loadings. Same logic as in scikit-learn's TransformerMixin method.

Parameters

- **t** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – T scores corresponding to the X data matrix.
- **u** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – Y scores corresponding to the Y data vector/matrix.

Return x X Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Return y Y Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Raises **ValueError** – If dimensions of input data are mismatched.

score(*x*, *y*, *block_to_score='y'*, *sample_weight=None*)

Predict and calculate the R2 for the model using one of the data blocks (X or Y) provided. Equivalent to the scikit-learn RegressorMixin score method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **block_to_score** (*str*) – Which of the data blocks (X or Y) to calculate the R2 goodness of fit.
- **sample_weight** (*numpy.ndarray, shape [n_samples] or None*) – Optional sample weights to use in scoring.

Return R2Y The model's R2Y, calculated by predicting Y from X and scoring.

Return type float

Return R2X The model's R2X, calculated by predicting X from Y and scoring.

Return type float

Raises **ValueError** – If block to score argument is not acceptable or date mismatch issues with the provided data.

predict(*x=None*, *y=None*)

Predict the values in one data block using the other. Same as its scikit-learn's RegressorMixin namesake method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Predicted data block (X or Y) obtained from the other data block.

Return type numpy.ndarray, shape [n_samples, n_features]

Raises

- **ValueError** – If no data matrix is passed, or dimensions mismatch issues with the provided data.
- **AttributeError** – Calling the method without fitting the model before.

VIP(*mode='w'*, *direction='y'*)

Output the Variable importance for projection metric (VIP). With the default values it is calculated using the x variable weights and the variance explained of y.

Note: Code not adequate to obtain a VIP for each individual variable in the multi-Y case, as SSY should be changed so that it is calculated for each y and not for the whole Y matrix

Parameters

- **mode** (*str*) – The type of model parameter to use in calculating the VIP. Default value is weights (w), and other acceptable arguments are p, ws, cs, c and q.
- **direction** (*str*) – The data block to be used to calculated the model fit and regression sum of squares.

Return numpy.ndarray VIP The vector with the calculated VIP values.

Return type numpy.ndarray, shape [n_features]

Raises

- **ValueError** – If mode or direction is not a valid option.
- **AttributeError** – Calling method without a fitted model.

hotelling_T2(*comps=[0, 1]*, *alpha=0.05*)

Obtain the parameters for the Hotelling T2 ellipse at the desired significance level.

Parameters

- **comps** (*list*) – List of components to calculate the Hotelling T2.
- **alpha** (*float*) – Significant level for the F statistic.

Returns List with the Hotelling T2 ellipse radii

Return type list

Raises **ValueError** – If the dimensions request

dmodx(*x*)

Normalised DmodX measure

Parameters **x** – data matrix [n samples, m variables]

Returns The Normalised DmodX measure for each sample

leverages(*block='X'*)

Calculate the leverages for each observation :return: :rtype:

outlier(*x*, *comps=None*, *measure='T2'*, *alpha=0.05*)

Use the Hotelling T2 or DmodX measure and F statistic to screen for outlier candidates.

Parameters

- **x** – Data matrix [n samples, m variables]

- **comps** – Which components to use (for Hotelling T2 only)
- **measure** – Hotelling T2 or DmodX
- **alpha** – Significance level

Returns List with row indices of X matrix

cross_validation(*x*, *y*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, *outputdist=False*, ***crossval_kwargs*)

Cross-validation method for the model. Calculates Q2 and cross-validated estimates for all model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.
- **outputdist** (*bool*) – Output the whole distribution for. Useful when ShuffleSplit or CrossValidators other than KFold.
- **crossval_kwargs** (*kwargs*) – Keyword arguments to be passed to the sklearn.Pipeline during cross-validation

Returns

Return type dict

Raises

- **TypeError** – If the cv_method passed is not a scikit-learn CrossValidator object.
- **ValueError** – If the x and y data matrices are invalid.

permutation_test(*x*, *y*, *nperms=1000*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, ***permtest_kwargs*)

Permutation test for the classifier. Outputs permuted null distributions for model performance metrics (Q2X/Q2Y) and most model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **nperms** (*int*) – Number of permutations to perform.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.
- **permtest_kwargs** (*kwargs*) – Keyword arguments to be passed to the .fit() method during cross-validation and model fitting.

Returns Permuted null distributions for model parameters and the permutation p-value for the Q2Y value.

Return type dict

2.3 ChemometricsPLSDA

class pyChemometrics.**ChemometricsPLSDA**(*ncomps=2*, *pls_algorithm=<class 'sklearn.cross_decomposition._pls.PLSRegression'>*, *xscaler=ChemometricsScaler()*, ***pls_type_kwargs*)

Chemometrics PLS-DA object - Similar to ChemometricsPLS, but with extra functions to handle Y vectors encoding class membership and classification assessment metrics.

Parameters

- **ncomps** (*int*) – Number of PLS components desired.
- **pls_algorithm** (*sklearn._PLS*) – Scikit-learn PLS algorithm to use - PLSRegression or PLSCanonical are supported.
- **xscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for X data matrix.
- **yscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for the Y data vector/matrix.
- **pls_type_kwargs** (*kwargs*) – Keyword arguments to be passed during initialization of pls_algorithm.

Raises **TypeError** – If the pca_algorithm or scaler objects are not of the right class.

fit(*x*, *y*, ***fit_params*)

Perform model fitting on the provided x and y data and calculate basic goodness-of-fit metrics. Similar to scikit-learn's BaseEstimator method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Keyword arguments to be passed to the .fit() method of the core sklearn model.

Raises **ValueError** – If any problem occurs during fitting.

fit_transform(*x*, *y*, ***fit_params*)

Fit a model to supplied data and return the scores. Equivalent to scikit-learn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Optional keyword arguments to be passed to the pls_algorithm .fit() method.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple of numpy.ndarray, shape [[n_tscores], [n_uscores]]

Raises **ValueError** – If any problem occurs during fitting.

transform(*x=None*, *y=None*)
Calculate the scores for a data block from the original data. Equivalent to sklearn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple with 2 numpy.ndarray, shape [n_samples, n_comps]

Raises

- **ValueError** – If dimensions of input data are mismatched.
- **AttributeError** – When calling the method before the model is fitted.

inverse_transform(*t=None*, *u=None*)
Transform scores to the original data space using their corresponding loadings. Same logic as in scikit-learn's TransformerMixin method.

Parameters

- **t** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – T scores corresponding to the X data matrix.
- **u** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – Y scores corresponding to the Y data vector/matrix.

Return x X Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Return y Y Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Raises **ValueError** – If dimensions of input data are mismatched.

score(*x*, *y*, *sample_weight=None*)
Predict and calculate the R2 for the model using one of the data blocks (X or Y) provided. Equivalent to the scikit-learn ClassifierMixin score method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **block_to_score** (*str*) – Which of the data blocks (X or Y) to calculate the R2 goodness of fit.
- **sample_weight** (*numpy.ndarray, shape [n_samples] or None*) – Optional sample weights to use in scoring.

Return R2Y The model's R2Y, calculated by predicting Y from X and scoring.

Return type float

Return R2X The model's R2X, calculated by predicting X from Y and scoring.

Return type float

Raises **ValueError** – If block to score argument is not acceptable or date mismatch issues with the provided data.

predict(*x*)

Predict the values in one data block using the other. Same as its scikit-learn's RegressorMixin namesake method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Predicted data block (X or Y) obtained from the other data block.

Return type numpy.ndarray, shape [n_samples, n_features]

Raises

- **ValueError** – If no data matrix is passed, or dimensions mismatch issues with the provided data.
- **AttributeError** – Calling the method without fitting the model before.

VIP(*mode='w'*, *direction='y'*)

Output the Variable importance for projection metric (VIP). With the default values it is calculated using the x variable weights and the variance explained of y. Default mode is recommended (mode = 'w' and direction = 'y')

Parameters

- **mode** (*str*) – The type of model parameter to use in calculating the VIP. Default value is weights (w), and other acceptable arguments are p, ws, cs, c and q.
- **direction** (*str*) – The data block to be used to calculated the model fit and regression sum of squares.

Return numpy.ndarray VIP The vector with the calculated VIP values.

Return type numpy.ndarray, shape [n_features]

Raises

- **ValueError** – If mode or direction is not a valid option.
- **AttributeError** – Calling method without a fitted model.

cross_validation(*x*, *y*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, *outputdist=False*, ***crossval_kwargs*)

Cross-validation method for the model. Calculates Q2 and cross-validated estimates for all model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.

- **outputdist** (*bool*) – Output the whole distribution for. Useful when ShuffleSplit or CrossValidators other than KFold.
- **crossval_kwargs** (*kwargs*) – Keyword arguments to be passed to the sklearn.Pipeline during cross-validation

Returns

Return type dict

Raises

- **TypeError** – If the cv_method passed is not a scikit-learn CrossValidator object.
- **ValueError** – If the x and y data matrices are invalid.

permutation_test(*x*, *y*, *nperms=1000*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, ***permtest_kwargs*)

Permutation test for the classifier. Outputs permuted null distributions for model performance metrics (Q2X/Q2Y) and many other model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **nperms** (*int*) – Number of permutations to perform.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.
- **permtest_kwargs** (*kwargs*) – Keyword arguments to be passed to the .fit() method during cross-validation and model fitting.

Returns Permuted null distributions for model parameters and the permutation p-value for the Q2Y value.

Return type dict

2.4 ChemometricsPLS_Logistic

class pyChemometrics.**ChemometricsPLS_Logistic**(*ncomps=2*, *pls_algorithm=<class 'sklearn.cross_decomposition._pls.PLSRegression'>*, *logreg_algorithm=<class 'sklearn.linear_model._logistic.LogisticRegression'>*, *xscaler=ChemometricsScaler()*, ***pls_type_kwargs*)

ChemometricsPLS object - Wrapper for sklearn.cross_decomposition PLS algorithms, with tailored methods for Chemometric Data analysis.

Parameters

- **ncomps** (*int*) – Number of PLS components desired.
- **pls_algorithm** (*sklearn._PLS*) – Scikit-learn PLS algorithm to use - PLSRegression or PLSCanonical are supported.
- **xscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for X data matrix.

- **yscaler** (*ChemometricsScaler object, scaling/preprocessing objects from scikit-learn or None.*) – Scaler object for the Y data vector/matrix.
- **pls_type_kwargs** (*kwargs*) – Keyword arguments to be passed during initialization of pls_algorithm.

Raises **TypeError** – If the pca_algorithm or scaler objects are not of the right class.

fit (*x*, *y*, ***fit_params*)
Perform model fitting on the provided x and y data and calculate basic goodness-of-fit metrics. Similar to scikit-learn's BaseEstimator method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Keyword arguments to be passed to the .fit() method of the core sklearn model.

Raises **ValueError** – If any problem occurs during fitting.

fit_transform (*x*, *y*, ***fit_params*)
Fit a model to supplied data and return the scores. Equivalent to scikit-learn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **fit_params** (*kwargs*) – Optional keyword arguments to be passed to the pls_algorithm .fit() method.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple of numpy.ndarray, shape [[n_tscores], [n_uscores]]

Raises **ValueError** – If any problem occurs during fitting.

transform (*x=None*, *y=None*)
Calculate the scores for a data block from the original data. Equivalent to sklearn's TransformerMixin method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Latent Variable scores (T) for the X matrix and for the Y vector/matrix (U).

Return type tuple with 2 numpy.ndarray, shape [n_samples, n_comps]

Raises

- **ValueError** – If dimensions of input data are mismatched.

- **AttributeError** – When calling the method before the model is fitted.

inverse_transform(*t=None*, *u=None*)

Transform scores to the original data space using their corresponding loadings. Same logic as in scikit-learn's TransformerMixin method.

Parameters

- **t** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – T scores corresponding to the X data matrix.
- **u** (*numpy.ndarray, shape [n_samples, n_comps] or None*) – Y scores corresponding to the Y data vector/matrix.

Return x X Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Return y Y Data matrix in the original data space.

Return type numpy.ndarray, shape [n_samples, n_features] or None

Raises **ValueError** – If dimensions of input data are mismatched.

score(*x*, *y*, *sample_weight=None*)

Predict and calculate the R2 for the model using one of the data blocks (X or Y) provided. Equivalent to the scikit-learn ClassifierMixin score method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **block_to_score** (*str*) – Which of the data blocks (X or Y) to calculate the R2 goodness of fit.
- **sample_weight** (*numpy.ndarray, shape [n_samples] or None*) – Optional sample weights to use in scoring.

Return R2Y The model's R2Y, calculated by predicting Y from X and scoring.

Return type float

Return R2X The model's R2X, calculated by predicting X from Y and scoring.

Return type float

Raises **ValueError** – If block to score argument is not acceptable or date mismatch issues with the provided data.

predict(*x*)

Predict the values in one data block using the other. Same as its scikit-learn's RegressorMixin namesake method.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features] or None*) – Data matrix to fit the PLS model.

Returns Predicted data block (X or Y) obtained from the other data block.

Return type numpy.ndarray, shape [n_samples, n_features]

Raises

- **ValueError** – If no data matrix is passed, or dimensions mismatch issues with the provided data.
- **AttributeError** – Calling the method without fitting the model before.

VIP(*mode='w'*, *direction='y'*)

Output the Variable importance for projection metric (VIP). With the default values it is calculated using the x variable weights and the variance explained of y.

Parameters

- **mode** (*str*) – The type of model parameter to use in calculating the VIP. Default value is weights (w), and other acceptable arguments are p, ws, cs, c and q.
- **direction** (*str*) – The data block to be used to calculated the model fit and regression sum of squares.

Return numpy.ndarray VIP The vector with the calculated VIP values.

Return type numpy.ndarray, shape [n_features]

Raises

- **ValueError** – If mode or direction is not a valid option.
- **AttributeError** – Calling method without a fitted model.

cross_validation(*x*, *y*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, *outputdist=False*, ***crossval_kwargs*)

Cross-validation method for the model. Calculates Q2 and cross-validated estimates for all model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.
- **outputdist** (*bool*) – Output the whole distribution for. Useful when ShuffleSplit or CrossValidators other than KFold.
- **crossval_kwargs** (*kwargs*) – Keyword arguments to be passed to the sklearn.Pipeline during cross-validation

Returns

Return type dict

Raises

- **TypeError** – If the cv_method passed is not a scikit-learn CrossValidator object.
- **ValueError** – If the x and y data matrices are invalid.

permutation_test(*x*, *y*, *nperms=1000*, *cv_method=KFold(n_splits=7*, *random_state=None*, *shuffle=True)*, ***permtest_kwargs*)

Permutation test for the classifier. Outputs permuted null distributions for model performance metrics (Q2X/Q2Y) and most model parameters.

Parameters

- **x** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **y** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to fit the PLS model.
- **nperms** (*int*) – Number of permutations to perform.
- **cv_method** (*BaseCrossValidator or BaseShuffleSplit*) – An instance of a scikit-learn CrossValidator object.
- **permtest_kwargs** (*kwargs*) – Keyword arguments to be passed to the .fit() method during cross-validation and model fitting.

Returns Permuted null distributions for model parameters and the permutation p-value for the Q2Y value.

Return type dict

2.5 ChemometricsScaler

class pyChemometrics.**ChemometricsScaler**(*scale_power=1*, *copy=True*, *with_mean=True*, *with_std=True*)

Extension of Scikit-learn's StandardScaler which allows scaling by different powers of the standard deviation.

Parameters

- **scale_power** (*Float*) – To which power should the standard deviation of each variable be raised for scaling. 0: Mean centering; 0.5: Pareto; 1:Unit Variance.
- **copy** (*bool*) – Copy the array containing the data.
- **with_mean** (*bool*) – Perform mean centering.
- **with_std** (*bool*) – Scale the data.

fit(*X*, *y=None*)

Compute the mean and standard deviation from a dataset to use in future scaling operations.

Parameters

- **X** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to scale.
- **y** (*None*) – Passthrough for Scikit-learn Pipeline compatibility.

Returns Fitted object.

Return type *pyChemometrics.ChemometricsScaler*

partial_fit(*X*, *y=None*)

Performs online computation of mean and standard deviation on X for later scaling. All of X is processed as a single batch. This is intended for cases when *fit* is not feasible due to very large number of *n_samples* or because X is read from a continuous stream.

The algorithm for incremental mean and std is given in Equation 1.5a,b in Chan, Tony F., Gene H. Golub, and Randall J. LeVeque. "Algorithms for computing the sample variance: Analysis and recommendations." The American Statistician 37.3 (1983): 242-247

Parameters

- **X** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to scale.
- **y** (*None*) – Passthrough for Scikit-learn `Pipeline` compatibility.

Returns Fitted object.

Return type *pyChemometrics.ChemometricsScaler*

transform(*X*, *y=None*, *copy=None*)

Perform standardization by centering and scaling using the parameters.

Parameters

- **X** (*numpy.ndarray, shape [n_samples, n_features]*) – Data matrix to scale.
- **y** (*None*) – Passthrough for scikit-learn `Pipeline` compatibility.
- **copy** (*bool*) – Copy the X matrix.

Returns Scaled version of the X data matrix.

Return type numpy.ndarray, shape [n_samples, n_features]

inverse_transform(*X*, *copy=None*)

Scale back the data to the original representation.

Parameters

- **X** (*numpy.ndarray, shape [n_samples, n_features]*) – Scaled data matrix.
- **copy** (*bool*) – Copy the X data matrix.

Returns X data matrix with the scaling operation reverted.

Return type numpy.ndarray, shape [n_samples, n_features]

CHAPTER 3

Using the pyChemometrics Models

The *tutorial* introduces the main objects, exemplified with typical use cases.

CHAPTER 4

The pyChemometrics object classes

A detailed description of the main classes and their methods included in the package is available in *pyChemometrics objects*.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

S

T

V

X