PLT Documentation

Release 0.2.3

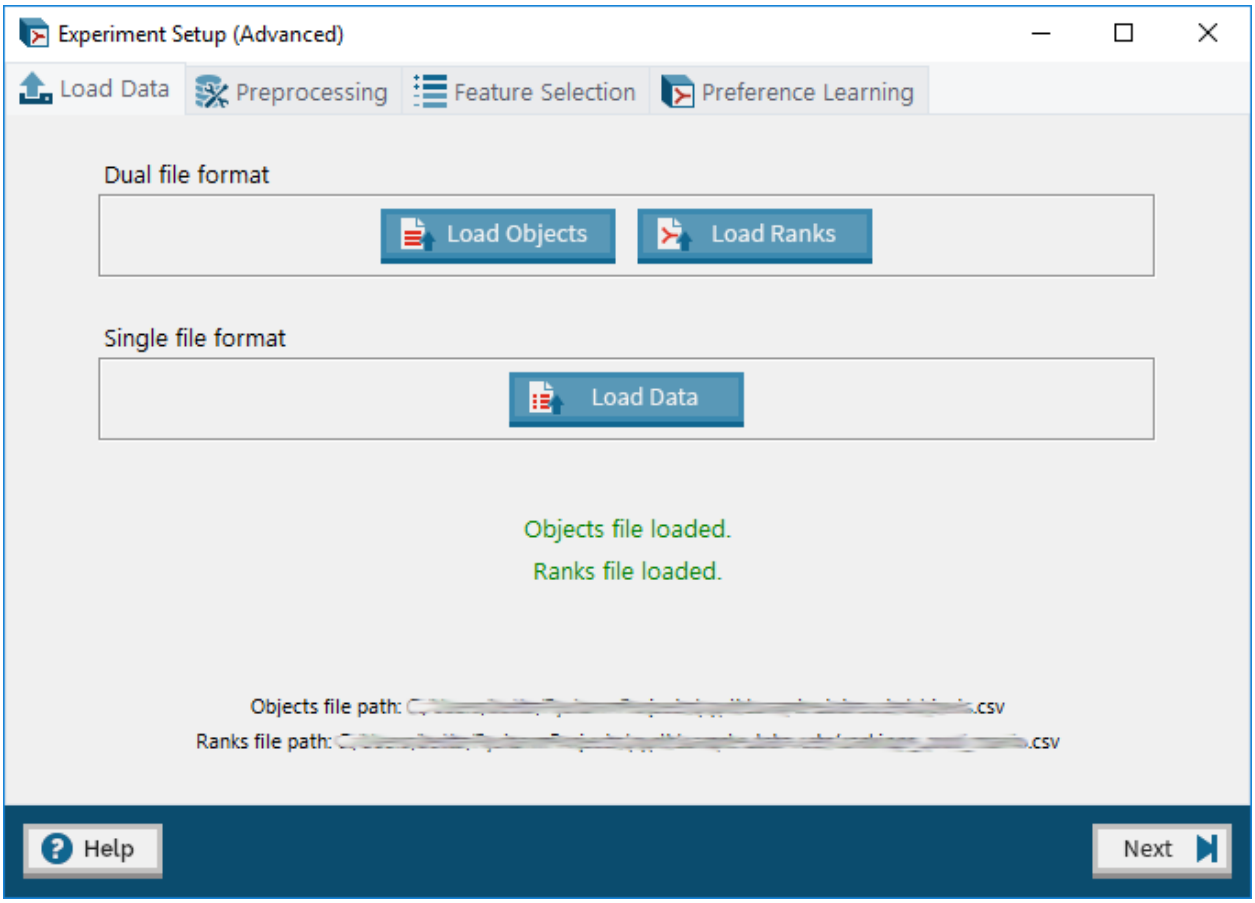
Institute of Digital Games (University of Malta)

Apr 10, 2019

Contents

Preference learning (PL) is a core area of machine learning that handles datasets with ordinal relations. As the number of generated data of ordinal nature such as ranks and subjective ratings is increasing, the importance and role of the PL field becomes central within machine learning research and practice.

The Preference Learning Toolbox (PLT) is an open source software application and package which supports the key data modelling phases incorporating various popular data pre-processing, feature selection and preference learning methods.



CHAPTER 1

PLT Features:

- Dataset Pre-processing (including automatic feature extraction)
- Automatic Feature Selection (SFS)
- Preference Learning Algorithms (RankSVM, ANN-Backpropagation, RankNet)
- Experiment Reporting and Model Storage

CHAPTER 2

Contents:

The documentation for PLT consists of the following:

2.1 API Reference

The **Preference Learning Toolbox** (**PLT**) is an open source software application and package which supports the key data modelling phases incorporating various popular data pre-processing, feature selection and preference learning methods.

This documentation describes the API of the **PLT** in detail.

2.1.1 Module contents

The graphical user interface (GUI) component of PLT is managed by the *pyplt.gui* subpackage and run via the *pyplt.main_gui* script. The remaining submodules and subpackages within this package focus on the backend functionality and the application programming interface (API) component of PLT.

2.1.2 Subpackages

pyplt.evaluation package

Submodules

pyplt.evaluation.base module

class pyplt.evaluation.base.**Evaluator**(*description='A validation/testing method.'*, *name=''*, *debug=False*, ***kwargs*)

Bases: `object`

Base class for all evaluation (validation or testing) methods.

Initializes the Evaluator object.

Parameters

- **description** (*str, optional*) – a description of the evaluation method (default "A validation/testing method.").
- **name** (*str, optional*) – the name of the evaluation method (default "").
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging purposes (default False).
- **kwargs** – any additional parameters for the evaluation method.

get_description()
Get the description of the evaluation method.

Returns the description of the evaluation method.

Return type str

get_name()
Get the name of the evaluation method.

Returns the name of the evaluation method.

Return type str

get_params()
Return all additional parameters of the evaluation method (if applicable).

Returns a dict containing all additional parameters of the evaluation method with the parameter names as the dict's keys and the corresponding parameter values as the dict's values (if applicable).

Return type dict

get_params_string()
Return a string representation of all additional parameters of the evaluation method (if applicable).

Returns the string representation of all additional parameters of the evaluation method (if applicable).

Return type str

pyplt.evaluation.cross_validation module

class pyplt.evaluation.cross_validation.**KFoldCrossValidation**(*k=3*, *test_folds=None*)

Bases: *pyplt.evaluation.base.Evaluator*

K-Fold Cross Validation.

Initializes the KFoldCrossValidation object.

The dataset may be split into folds in two ways: automatically or manually. If automatic, the *k* argument is to be used. If manual, the user may specify the fold index for each sample in the dataset via the *test_folds* argument.

Parameters

- **k** (*int, optional*) – the number of folds to uniformly split the data into when using the automatic approach (default 3).

- **test_folds** (*numpy.ndarray* or None, optional) – an array specifying the fold index for each sample in the dataset when using the manual approach (default None). The entry test_folds[i] specifies the index of the test set that sample i belongs to. It is also possible to exclude sample i from any test set (i.e., include sample i in every training set) by setting test_folds[i] to -1. If *test_folds* is None, the automatic approach is assumed and only the *k* parameter is considered. Otherwise, the manual approach is assumed and only the *test_folds* parameter is considered.

Raises ***InvalidParameterValueException*** – if a *k* parameter value less than 2 is used.

split(*data*)

Get the indices for the training set and test set of the next fold of the dataset.

If the single file format is used, the indices are given with respect to objects. Otherwise (if the dual file format is used), the indices are given with respect to the ranks.

Parameters **data** (*pandas.DataFrame* or tuple of *pandas.DataFrame* (size 2)) – the data to be split into folds. If the single file format is used, a single *pandas.DataFrame* containing the data should be passed. If the dual file format is used, a tuple containing both the objects and ranks (each a *pandas.DataFrame*) should be passed.

Returns **yields** two arrays containing the integer-based indices for the training set and test set of the next fold.

Return type

- train: *numpy.ndarray*
- test: *numpy.ndarray*

class pyplt.evaluation.cross_validation.**PreprocessedFolds**(*folds*)

Bases: object

Class for neatly storing and working with a dataset that has been split into two or more folds.

The data in each fold is assumed to be pre-processed prior to instantiation of this class.

Initializes the *PreprocessedFolds* instance and stores the given fold data.

Parameters **folds** (*list of tuples (size 4):*) – a list of tuples containing the pre-processed training set and test set (if applicable) of each fold. Each tuple (fold) should contain:

- train_objects: *pandas.DataFrame*
- train_ranks: *pandas.DataFrame*
- test_objects: *pandas.DataFrame* (if applicable) or None
- test_ranks: *pandas.DataFrame* (if applicable) or None

If either the *test_objects* or *test_ranks* of the first fold is None, it is assumed that only training will be carried out.

get_features()

Get the features defining the objects in the data.

These are determined by looking at the features of the training objects in the first fold.

get_n_folds()

Get the number of folds in the data.

is_training_only()

Indicate whether or not training only is to be applied on the given data.

next_fold()
Get the pre-processed training set and test set (if applicable) of the next fold.

Returns **yields** the pre-processed training set and test set (if applicable) of the next fold.

Return type

- train_objects: *pandas.DataFrame*
- train_ranks: *pandas.DataFrame*
- test_objects: *pandas.DataFrame* (if applicable) or None
- test_ranks: *pandas.DataFrame* (if applicable) or None

pyplt.evaluation.holdout module

class pyplt.evaluation.holdout.**HoldOut**(*test_proportion=0.3*, *debug=False*)
Bases: *pyplt.evaluation.base.Evaluator*

Holdout evaluator.

This evaluation method splits the pairwise rank data into a training set and a test set. The training set is used to train the model via preference learning whereas the test set is used to estimate the prediction accuracy of the model. Often, 70% of the data is used as the training set while the remaining 30% is used as the test set (i.e., a test proportion of 0.3) however the user may choose a different proportion.

Initializes the HoldOut object.

Parameters

- **test_proportion** (*float, optional*) – the proportion of data to be used as the test set; the remaining data is used as the training data (default 0.3).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).

split(*data*)
Split the given dataset into a training set and a test set according to the given proportion parameter.

If the single file format is used, the indices are given with respect to objects. Otherwise (if the dual file format is used), the indices are given with respect to the ranks.

Parameters **data** (*pandas.DataFrame* or tuple of *pandas.DataFrame* (size 2)) – the data to be split into folds. If the single file format is used, a single *pandas.DataFrame* containing the data should be passed. If the dual file format is used, a tuple containing both the objects and ranks (each a *pandas.DataFrame*) should be passed.

Returns two arrays containing the indices for the training set and test set.

Return type

- train: *numpy.ndarray*
- test: *numpy.ndarray*

Module contents

This package contains backend modules that manage the evaluation step of an experiment.

pyplt.fsmethods package

Submodules

pyplt.fsmethods.base module

class pyplt.fsmethods.base.**FeatureSelectionMethod**(*description='A feature selection method.'*, *name=''*, ***kwargs*)

Bases: object

Base class for all feature selection methods.

Initializes the FeatureSelectionMethod object.

Parameters

- **description** (*str, optional*) – a description of the feature selection method (default "A feature selection method.").
- **name** (*str, optional*) – the name of the feature selection method (default "").
- **kwargs** – any additional parameters for the feature selection method.

get_description()
Get the description of the feature selection method.

Returns the description of the feature selection method.

Return type str

get_name()
Get the name of the feature selection method.

Returns the name of the feature selection method.

Return type str

get_params()
Return all additional parameters of the feature selection method (if applicable).

Returns a dict containing all additional parameters of the feature selection method with the parameter names as the dict's keys and the corresponding parameter values as the dict's values (if applicable).

Return type dict

get_params_string()
Return a string representation of all additional parameters of the feature selection method (if applicable).

Returns the string representation of all additional parameters of the feature selection method (if applicable).

Return type str

get_selected_features()
Get the subset of selected features.

Returns the subset of selected features.

Return type list of str

select(*objects*, *ranks*, *algorithm*, *test_objects=None*, *test_ranks=None*, *preprocessed_folds=None*, *progress_window=None*, *exec_stopper=None*)
Abstract method for running the feature selection process.

All children classes must implement this method.

Parameters

- **objects** (*pandas.DataFrame* or None) – the objects data to be used during the feature selection process. If None, the data is obtained via the *preprocessed_folds* parameter instead.
- **ranks** (*pandas.DataFrame* or None) – the pairwise rank data to be used during the feature selection process. If None, the data is obtained via the *preprocessed_folds* parameter instead.
- **algorithm** (*pyplt.plalgorithms.base.PLAlgorithm*) – the algorithm to be used for feature selection (if applicable).
- **test_objects** (*pandas.DataFrame* or None, optional) – optional test objects data to be used during the feature selection process (default None).
- **test_ranks** (*pandas.DataFrame* or None, optional) – optional test pairwise rank data to be used during the feature selection process (default None).
- **preprocessed_folds** (*pyplt.evaluation.cross_validation.PreprocessedFolds* or None, optional) – the data used to evaluate the feature set with in the form of pre-processed folds (default None). This is an alternative way to pass the data and is only considered if either of the *objects* and *ranks* parameters is None.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the subset of selected features – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type list of str

pyplt.fsmethods.sfs module

class pyplt.fsmethods.sfs.**SFS**(*verbose=True*)

Bases: *pyplt.fsmethods.wrappers.WrapperFSMethod*

Sequential Forward Selection (SFS) method.

SFS is a bottom-up hill-climbing algorithm where one feature is added at a time to the current feature set. The feature to be added is selected from the subset of the remaining features such that the new feature set generates the maximum value of the performance function over all candidate features for addition. The selection procedure begins with an empty feature set and terminates when an added feature yields equal or lower performance to the performance obtained without it. The performance of each subset of features considered is computed as the prediction accuracy of a model trained using that subset of features as input. All of the preference learning algorithms implemented in the tool can be used to train this model; i.e., RankSVM and Backpropagation.

Extends the *pyplt.fsmethods.wrappers.WrapperFSMethod* class which, in turn, extends the *pyplt.fsmethods.base.FeatureSelectionMethod* class.

Initializes the feature selection method with the appropriate name and description.

Parameters **verbose** (*bool*) – specifies whether or not to display detailed progress information to the *progress_window* if one is used (default True).

select(*objects*, *ranks*, *algorithm*, *test_objects=None*, *test_ranks=None*, *preprocessed_folds=None*, *progress_window=None*, *exec_stopper=None*)
Carry out the feature selection process according to the SFS algorithm.

Parameters

- **objects** (*pandas.DataFrame* or None) – the objects data used to train the models used to evaluate and select features. If None, the data is obtained via the *preprocessed_folds* parameter instead.
- **ranks** (*pandas.DataFrame* or None) – the pairwise rank data used to train the models used to evaluate and select features. If None, the data is obtained via the *preprocessed_folds* parameter instead.
- **algorithm** (*pyplt.plalgorithms.base.PLAlgorithm*) – the algorithm used to train models to evaluate the features with (via the training accuracy).
- **test_objects** (*pandas.DataFrame* or None, optional) – optional objects data used to test the models used to evaluate and select features (default None).
- **test_ranks** (*pandas.DataFrame* or None, optional) – optional pairwise rank data used to test the models used to evaluate and select features (default None).
- **preprocessed_folds** (*pyplt.evaluation.cross_validation.PreprocessedFolds* or None, optional) – the data used to evaluate the feature set with in the form of pre-processed folds (default None). This is an alternative way to pass the data and is only considered if either of the *objects* and *ranks* parameters is None.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the subset of features selected by SFS – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type list of str

pyplt.fsmethods.wrappers module

class pyplt.fsmethods.wrappers.**WrapperFSMethod**(*description='A wrapper feature selection method.'*, ***kwargs*)
Bases: *pyplt.fsmethods.base.FeatureSelectionMethod*

Parent class for all wrapper-type feature selection methods.

Initializes the wrapper-type feature selection method.

Parameters

- **description** (*str, optional*) – a description of the feature selection method (default "A wrapper feature selection method.").
- **kwargs** – any additional parameters for the feature selection method.

Module contents

This package contains backend modules that manage the feature selection step of an experiment.

pyplt.gui package

Subpackages

pyplt.gui.experiment package

Subpackages

pyplt.gui.experiment.dataset package

Submodules

pyplt.gui.experiment.dataset.loading module

class pyplt.gui.experiment.dataset.loading.**DataLoadingTab**(*parent*, *parent_window*)

Bases: tkinter.Frame

GUI tab for the data set loading stage of setting up an experiment.

Extends the class *tkinter.Frame*.

Populates the frame widget.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this frame widget.

get_data()

Get the loaded data.

If the single file format is used, a single *pandas.DataFrame* containing the data is returned. If the dual file format is used, a tuple containing both the objects and ranks (each a *pandas.DataFrame*) is returned.

Returns the loaded data.

Return type *pandas.DataFrame* or tuple of *pandas.DataFrame* (size 2)

get_objects_path()

Get the objects file path, if applicable.

Returns the objects file path.

Return type str

get_rank_derivation_params()

Get the values of the rank derivation parameters (minimum distance margin and memory).

These only apply when a single file format is used.

Returns tuple containing the values of both parameters.

Return type tuple (size 2)

get_ranks_path()
Get the ranks file path, if applicable.

Returns the ranks file path.

Return type str

get_single_path()
Get single file path, if applicable.

Returns the single file path.

Return type str

is_data_loaded()
Check if a full data set has been loaded.

A full data set consists of either both a valid objects file and a valid and compatible ranks file or a valid single file.

Returns boolean indicating whether or not a full data set is loaded.

Return type bool

is_new_data()
One-time check to verify whether new data (which may override any previously loaded data) has been loaded.

If `self._new_data` is true, it is reset to False before this method returns True.

Returns boolean indicating whether or not new data has been loaded (i.e., True if self._new_data is True; False otherwise).

Return type bool

pyplt.gui.experiment.dataset.params module

class pyplt.gui.experiment.dataset.params.**Confirmation**
Bases: `enum.Enum`

Class specifying enumerated constants for loading confirmation states.

Extends *enum.Enum*.

CANCEL = 0

DONE = 1

class pyplt.gui.experiment.dataset.params.**LoadingFoldsWindow**(*parent*, *file_path*, *parent_window*, *is_dual_format*)
Bases: `tkinter.Toplevel`

GUI window for specifying the parameters for loading data from a file.

Extends the class *tkinter.Toplevel*.

Populates the window widget with widgets to specify loading parameters and a data set preview frame.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this window widget.
- **file_path** (*str*) – the path of the file to be loaded.

- **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.
- **is_dual_format** (*bool*) – specifies whether the dual file format was used to load the dataset or not (single file format).

get_confirmation()
Get the confirmation variable indicating whether or not the file load has been completed or cancelled.

Returns the confirmation variable for the file that was attempted to be loaded.

Return type *pyplt.gui.experiment.dataset.params.Confirmation*

get_data()
Get the data extracted and processed from the file being loaded.

Returns the extracted data.

Return type *pandas.DataFrame*

class pyplt.gui.experiment.dataset.params.**LoadingParamsWindow**(*parent*, *file_path*, *parent_window*, *file_type*)

Bases: tkinter.Toplevel

GUI window for specifying the parameters for loading data from a file.

Extends the class *tkinter.Toplevel*.

Populates the window widget with widgets to specify loading parameters and a data set preview frame.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this window widget.
- **file_path** (*str*) – the path of the file to be loaded.
- **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.
- **file_type** (*pyplt.util.enums.FileType*.) – the type of file to be loaded.

get_confirmation()
Get the confirmation variable indicating whether or not the file load has been completed or cancelled.

Returns the confirmation variable for the file that was attempted to be loaded.

Return type *pyplt.gui.experiment.dataset.params.Confirmation*

get_data()
Get the data extracted and processed from the file being loaded.

Returns the extracted data.

Return type *pandas.DataFrame*

get_rank_derivation_params()
Get the values of the rank derivation parameters (minimum distance margin and memory).

These only apply when a single file format is used.

Returns tuple containing the values of both parameters.

Return type tuple (size 2)

pyplt.gui.experiment.dataset.preview module

class pyplt.gui.experiment.dataset.preview.**DataSetPreviewFrame**(*master*, *df*, *heads_bg_colour='#106690'*, *heads_text_colour='white'*, *style_prefix=''*)

Bases: `object`

GUI frame for previewing datasets loaded into PLT.

Instantiates a *tkinter.Canvas* object containing the frame previewing the data.

Parameters

- **master** (*tkinter widget*) – the parent widget of the data preview canvas widget.
- **df** (*pandas.DataFrame*) – the data to be previewed.
- **heads_bg_colour** (*str, optional*) – the background colour to be used for the column headers (representing feature names) in the preview (default `pyplt.gui.util.colours.PREVIEW_DEFAULT`).
- **heads_text_colour** (*str, optional*) – the text colour to be used for the column headers (representing feature names) in the preview (default 'white').
- **style_prefix** (*str, optional*) – specifies an additional prefix to add to the name of the style used for themed *ttk widgets* (should include a dot character at the end) (default '').

destroy_preview()
Destroy the preview frame widget and the canvas widget containing it.

update(*df*)
Update the preview frame with the given data.

Parameters **df** (*pandas.DataFrame*) – the data to be previewed.

update_column(*col_id*, *new_values*)
Update the values of a single column in the preview (e.g., for normalization).

Parameters

- **col_id** (*int*) – the index of the column to be updated.
- **new_values** (*array-like*) – the new values to be displayed in the given column.

Module contents

This package contains GUI-based modules that manage the data set loading stage of setting up an experiment.

pyplt.gui.experiment.featureselection package

Submodules

pyplt.gui.experiment.featureselection.featselectiontab module

class pyplt.gui.experiment.featureselection.featselectiontab.**FeatureSelectionFrame**(*parent*, *parent_window*, *files_tab*)

Bases: tkinter.Frame

Frame widget that is visible whenever the *Feature Selection* tab is in the 'unlocked' state.

Extends the class *tkinter.Frame*.

Initializes the frame widget.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this frame widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.

get_algorithm()
Get the preference learning algorithm type chosen by the user (if applicable).

Returns the preference learning algorithm type chosen by the user (if applicable).

Return type *pyplt.util.enums.PLAlgo*

get_algorithm_params()
Get the parameters of the preference learning algorithm chosen by the user (if applicable).

Returns the parameters of the preference learning algorithm chosen by the user (if applicable).

Return type list

get_evaluator()
Get the evaluation method type chosen by the user (if applicable).

Returns the evaluation method type chosen by the user (if applicable).

Return type *pyplt.util.enums.EvaluatorType*

get_evaluator_params()
Get the parameters of the evaluation method chosen by the user (if applicable).

Returns the parameters of the evaluation method chosen by the user (if applicable).

Return type list

get_method()
Get the feature selection method type chosen by the user.

Returns the feature selection method type chosen by the user.

Return type *pyplt.util.enums.FSMethod*

get_method_params()
Get the parameters of the feature selection method chosen by the user (if applicable).

Returns the parameters of the feature selection method chosen by the user (if applicable).

Return type list

class pyplt.gui.experiment.featureselection.featselectiontab.**FeatureSelectionTab**(*parent*, *parent_window*, *files_tab*)

Bases: *pyplt.gui.util.tab_locking.LockableTab*

GUI tab for the feature selection stage of setting up an experiment.

Extends the class *pyplt.gui.util.tab_locking.LockableTab* which, in turn, extends the *tkinter.Frame* class.

Initializes the *FeatureSelectionTab* object.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this tab (frame) widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this tab (frame) widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.

get_fs_algorithm()

Get the preference learning algorithm type chosen by the user via the *FeatureSelectionFrame* (if applicable).

Returns the preference learning algorithm type chosen by the user (if applicable).

Return type *pyplt.util.enums.PLAlgo*

get_fs_algorithm_params()

Get the parameters of the preference learning algorithm chosen by the user (if applicable).

Returns the parameters of the preference learning algorithm chosen by the user (if applicable).

Return type list

get_fs_evaluator()

Get the evaluation method type chosen by the user via the *FeatureSelectionFrame* (if applicable).

Returns the evaluation method type chosen by the user (if applicable).

Return type *pyplt.util.enums.EvaluatorType*

get_fs_evaluator_params()

Get the parameters of the evaluation method chosen by the user (if applicable).

Returns the parameters of the evaluation method chosen by the user (if applicable).

Return type list

get_fs_method()

Get the feature selection method type chosen by the user via the *FeatureSelectionFrame*.

Returns the feature selection method type chosen by the user.

Return type *pyplt.util.enums.FSMethod*

get_fs_method_params()

Get the parameters of the feature selection method chosen by the user (if applicable).

Returns the parameters of the feature selection method chosen by the user (if applicable).

Return type list

get_normal_frame()
Return a *FeatureSelectionFrame* widget for when the tab is in the 'unlocked' state.

The *FeatureSelectionFrame* widget is instantiated only once on the first occasion that the tab is 'unlocked'.

Returns the *FeatureSelectionFrame* widget that is visible whenever the tab is in the 'unlocked' state.

Return type *pyplt.gui.experiment.featureselection.featselectiontab.FeatureSelectionFrame*

Module contents

This package contains GUI-based modules that manage the feature selection stage of setting up an experiment.

pyplt.gui.experiment.preflearning package

Submodules

pyplt.gui.experiment.preflearning.backprop_menu module

class pyplt.gui.experiment.preflearning.backprop_menu.**BackpropMenu**(*parent*, *on_resize_fn*)

Bases: tkinter.Frame

GUI menu for specifying parameters of the *Backpropagation* algorithm.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **on_resize_fn** (*function*) – the function called when the parent window is resized by the user. This is required by this class so that the window is resized accordingly whenever the widgets for a hidden layer are added to or removed from the *BackpropMenu*.

get_params()
Get the values for the *Backpropagation* algorithm parameters as specified by the user via the GUI.

The parameter values are returned in the form of a dict where the keys match the keywords of the arguments that would be passed to the corresponding *pyplt.plalgorithms.base.PLAlgorithm* constructor.

Returns

a dict containing the values for the following parameters in order:

- ann_topology: the topology of the neurons in the network
- learn_rate: the learning rate
- error_threshold: the error threshold
- epochs: the number of epochs
- activation_functions: the activation functions for each neuron layer in the network

Return type dict (size 7)

pyplt.gui.experiment.preflearning.evaluator_menus module

class pyplt.gui.experiment.preflearning.evaluator_menus.**HoldoutMenu**(*parent*)
Bases: tkinter.Frame

GUI menu for specifying parameters of the *Holdout* evaluation method.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters **parent** (*tkinter window*) – the parent widget of this frame widget.

get_params()
Get the values for the *Holdout* method parameters as specified by the user via the GUI.

The parameter values are returned in the form of a dict where the keys match the keywords of the arguments that would be passed to the corresponding *pyplt.util.enums.EvaluatorType* constructor.

Returns

a dict containing the values for the following parameters in order:

- test_proportion - the fractional proportion of data to be used as test data (the rest is to be used as training data) (default 0.3).

Return type dict (size 1) of float

class pyplt.gui.experiment.preflearning.evaluator_menus.**KFCVMenu**(*parent*, *parent_window*, *files_tab*, *on_resize_fn*)
Bases: tkinter.Frame

GUI menu for specifying parameters of the *KFoldCrossValidation* evaluation method.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this tab (frame) widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.
- **on_resize_fn** (*function*) – the function called when the parent window is resized by the user. This is required by this class so that the window is resized accordingly whenever widgets are added to or removed from the *KFCVMenu* procedurally.

get_params()
Get the values for the *K-Fold Cross Validation* method parameters as specified by the user via the GUI.

The parameter values are returned in the form of a dict where the keys match the keywords of the arguments that would be passed to the corresponding *pyplt.util.enums.EvaluatorType* constructor.

Returns

a dict containing the values for the following parameters in order:

- k - the number of folds to uniformly split the data into when using the automatic approach (default 3).

- test_folds - an array specifying the fold index for each sample in the dataset when using the manual approach (default None). The entry test_folds[i] specifies the index of the test set that sample i belongs to. It is also possible to exclude sample i from any test set (i.e., include sample i in every training set) by setting test_folds[i] to -1. If *test_folds* is None, the automatic approach is assumed and only the *k* parameter is to be considered. Otherwise, the manual approach is to be assumed and only the *test_folds* parameter is to be considered.

Return type dict (size 2) containing: * k - int or None * test_folds - *numpy.ndarray* or None

Raises ***MissingManualFoldsException*** – if the user chooses to specify folds manually for cross validation but fails to load the required file containing the fold IDs.

pyplt.gui.experiment.preflearning.pltab module

class pyplt.gui.experiment.preflearning.pltab.**PLFrame**(*parent*, *parent_window*, *files_tab*, *preproc_tab*, *fs_tab*)

Bases: `tkinter.Frame`

Frame widget that is visible whenever the *Preference Learning* tab is in the 'unlocked' state.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this frame widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.
- **preproc_tab** (*pyplt.gui.experiment.preprocessing.preproctab.PreProcessingTab*) – the *Preprocessing* tab.
- **fs_tab** (*pyplt.gui.experiment.featureselection.featselectiontab.FeatureSelectionTab*) – the *Feature Selection* tab.

get_algorithm()
Get the preference learning algorithm type chosen by the user.

Returns the preference learning algorithm chosen by the user.

Return type *pyplt.util.enums.PLAlgo*

get_algorithm_params()
Get the parameters of the preference learning algorithm chosen by the user (if applicable).

Returns the parameters of the preference learning algorithm chosen by the user (if applicable).

Return type list

get_evaluator()
Get the evaluation method type chosen by the user.

Returns the evaluation method type chosen by the user.

Return type *pyplt.util.enums.EvaluatorType*

get_evaluator_params()
Get the parameters of the evaluation method chosen by the user (if applicable).

Returns the parameters of the evaluation method chosen by the user (if applicable).

Return type list

run_exp()
Trigger the execution of the experiment in a separate thread from the main (GUI) thread.

A *pyplt.gui.experiment.progresswindow.ProgressWindow* widget is also initialized to keep a progress log and progress bar of the experiment execution process.

Threading is carried out using the *threading.Thread* class.

class pyplt.gui.experiment.preflearning.pltab.**PLTab**(*parent*, *parent_window*, *files_tab*, *preproc_tab*, *fs_tab*)
Bases: *pyplt.gui.util.tab_locking.LockableTab*

GUI tab for the preference learning and evaluation stage of setting up an experiment.

Extends the class *pyplt.gui.util.tab_locking.LockableTab* which, in turn, extends the *tkinter.Frame* class.

Initializes the *PLTab* object.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this tab (frame) widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this tab (frame) widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.
- **preproc_tab** (*pyplt.gui.experiment.preprocessing.preproctab.PreProcessingTab*) – the *Preprocessing* tab.
- **fs_tab** (*pyplt.gui.experiment.featureselection.featselectiontab.FeatureSelectionTab*) – the *Feature Selection* tab.

get_evaluator()
Get the evaluation method type chosen by the user via the *PLFrame*.

Returns the evaluation method type chosen by the user.

Return type *pyplt.util.enums.EvaluatorType*

get_evaluator_params()
Get the parameters of the evaluation method chosen by the user (if applicable).

Returns the parameters of the evaluation method chosen by the user (if applicable).

Return type list

get_normal_frame()
Return a *PLFrame* widget for when the tab is in the 'unlocked' state.

The *PLFrame* widget is instantiated only once on the first occasion that the tab is 'unlocked'.

Returns the *PLFrame* widget that is visible whenever the tab is in the 'unlocked' state.

Return type *pyplt.gui.experiment.preflearning.pltab.PLFrame*

get_pl_algorithm()
Get the preference learning algorithm type chosen by the user via the *PLFrame*.

Returns the preference learning algorithm type chosen by the user.

Return type *pyplt.util.enums.PLAlgo*

get_pl_algorithm_params()
Get the parameters of the preference learning algorithm chosen by the user (if applicable).

Returns the parameters of the preference learning algorithm chosen by the user (if applicable).

Return type list

run_experiment()
Call the method which triggers the execution of the experiment.

pyplt.gui.experiment.preflearning.ranknet_menu module

class pyplt.gui.experiment.preflearning.ranknet_menu.**RankNetMenu**(*parent*, *on_resize_fn*)
Bases: tkinter.Frame

GUI menu for specifying parameters of the *RankNet* algorithm.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **on_resize_fn** (*function*) – the function called when the parent window is resized by the user. This is required by this class so that the window is resized accordingly whenever the widgets for a hidden layer are added to or removed from the *BackpropMenu*.

get_params()
Get the values for the *Backpropagation* algorithm parameters as specified by the user via the GUI.

The parameter values are returned in the form of a dict where the keys match the keywords of the arguments that would be passed to the corresponding *pyplt.plalgorithms.base.PLAlgorithm* constructor.

Returns

a dict containing the values for the following parameters in order:

- ann_topology: the topology of the neurons in the network
- learn_rate: the learning rate
- epochs: the number of epochs
- hidden_activation_functions: the activation functions for each hidden layer in the network
- batch_size: the number of samples per gradient update.

Return type dict (size 7)

pyplt.gui.experiment.preflearning.ranksvm_menu module

class pyplt.gui.experiment.preflearning.ranksvm_menu.**RankSVMMenu**(*parent*)
Bases: tkinter.Frame

GUI menu for specifying parameters of the *RankSVM* algorithm.

Extends the class *tkinter.Frame*.

Initializes the frame widget and its contents.

Parameters **parent** (*tkinter widget*) – the parent widget of this frame widget.

get_params()
Get the values for the *RankSVM* algorithm parameters as specified by the user via the GUI.

The parameter values are returned in the form of a dict where the keys match the keywords of the arguments that would be passed to the corresponding *pyplt.plalgorithms.base.PLAlgorithm* constructor.

Returns

a dict containing the values for the following parameters in order:

- kernel: the kernel name
- gamma: the gamma kernel parameter value
- degree: the degree kernel parameter value

Return type dict (size 3)

Module contents

This package contains GUI-based modules that manage the preference learning stage of setting up an experiment.

pyplt.gui.experiment.preprocessing package

Submodules

pyplt.gui.experiment.preprocessing.preproctab module

class pyplt.gui.experiment.preprocessing.preproctab.**PreProcessingFrame**(*parent*, *parent_window*, *files_tab*)

Bases: tkinter.Frame

Frame widget that is visible whenever the *Data Pre-Processing tab* is in the 'unlocked' state.

Extends the class *tkinter.Frame*.

Initializes the frame widget.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this frame widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this frame widget.
- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.

auto_extract_enabled()
Indicate whether or not automatic feature selection (via autoencoder) has been chosen.

Returns specifies whether or not automatic feature selection (via autoencder) was chosen.

Return type bool

destroy_tab()
Destroy the contents of the tab.

get_autoencoder_menu()
Get the autoencoder GUI menu widget through which the parameter values selected by the user may be read.

Returns the autoencoder menu widget.

Return type *:class:pyplt.gui.experiment.preprocessing.data_compression.AutoencoderSettings*

get_include_settings()
Get the current include/exclude settings for each feature in the original objects data.

Returns a dict containing the feature indices as the dict's keys and booleans indicating whether the corresponding feature is to be included in (True) or excluded from (False) the experiment as the dict's values.

Return type dict of bool

get_norm_settings()
Get the normalization settings for each feature in the original objects data.

Returns a dict containing the feature indices as the dict's keys and enumerated constants of type *pyplt.util.enums.NormalizationType* indicating how the corresponding feature is to be normalized as the dict's values.

Return type dict of *pyplt.util.enums.NormalizationType*

get_shuffle_settings()
Get the settings chosen by the user with respect to shuffling the dataset.

Returns

- shuffle – specifies whether or not to shuffle the dataset at the start of the experiment execution.
- random_seed – optional seed used to shuffle the dataset.

Return type

- shuffle – bool
- random_seed – int or None

init_tab()
Initialize (or re-initialize) the contents of the tab.

class pyplt.gui.experiment.preprocessing.preproctab.**PreProcessingTab**(*parent*, *parent_window*, *files_tab*)

Bases: *pyplt.gui.util.tab_locking.LockableTab*

GUI tab for the data pre-processing stage of setting up an experiment.

Extends the class *pyplt.gui.util.tab_locking.LockableTab* which, in turn, extends the *tkinter.Frame* class.

Initializes the *PreProcessingTab* widget.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this tab (frame) widget.
- **parent_window** (*tkinter.Toplevel*) – the window which will contain this tab (frame) widget.

- **files_tab** (*pyplt.gui.experiment.dataset.loading.DataLoadingTab*) – the *Load Data* tab.

auto_extract_enabled()
Indicate whether or not automatic feature selection (via autoencoder) has been chosen.

Returns specifies whether or not automatic feature selection (via autoencder) was chosen.

Return type bool

get_autoencoder_menu()
Get the autoencoder GUI menu widget through which the parameter values selected by the user may be read.

Returns the autoencoder menu widget.

Return type *:class:pyplt.gui.experiment.preprocessing.data_compression.AutoencoderSettings*

get_include_settings()
Get the user settings for each feature indicating whether or not it is to be included in the experiment.

Returns a dict containing the feature names as the dict's keys and booleans indicating whether the corresponding feature is to be included in (True) or excluded from (False) the experiment as the dict's values.

Return type dict of bool

get_norm_settings()
Get the user settings for each feature indicating how it is to be normalized.

Returns a dict containing the feature names as the dict's keys and enumerated constants of type *pyplt.util.enums.NormalizationType* indicating how the corresponding feature is to be normalized as the dict's values.

Return type dict of *pyplt.util.enums.NormalizationType*

get_normal_frame()
Return a *PreProcessingFrame* widget for when the tab is in the 'unlocked' state.

The *PreProcessingFrame* widget is instantiated only once on the first occasion that the tab is 'unlocked'.

Returns the *PreProcessingFrame* widget that is visible whenever the tab is in the 'unlocked' state.

Return type *pyplt.gui.experiment.preprocessing.preproctab.PreProcessingFrame*

get_shuffle_settings()
Get the settings chosen by the user with respect to shuffling the dataset.

Returns

- shuffle – specifies whether or not to shuffle the dataset at the start of the experiment execution.
- random_seed – optional seed used to shuffle the dataset.

Return type

- shuffle – bool
- random_seed – int or None

lock()
Override method in parent class to destroy the tab contents each time it switches to the 'locked' state.

This is carried out by calling the *destroy_tab()* method of the *pyplt.gui.experiment.preprocessing.preproctab.PreProcessingFrame* class.

refresh()
Destroy and re-initialize the tab contents.

This is done by subsequent calls to the *destroy_tab()* and *init_tab()* methods of the *pyplt.gui.experiment.preprocessing.preproctab.PreProcessingFrame* class.

unlock()
Override method in parent class to re-initialize the tab contents each time it switches to the 'unlocked' state.

This is done to ensure the tab contents reflect the most recently loaded data set and is carried out by calling the *init_tab()* method of the *pyplt.gui.experiment.preprocessing.preproctab.PreProcessingFrame* class.

Module contents

This package contains GUI-based modules that manage the data pre-processing stage of setting up an experiment.

pyplt.gui.experiment.results package

Submodules

pyplt.gui.experiment.results.resultsscreen module

class pyplt.gui.experiment.results.resultsscreen.**ResultsWindow**(*parent*, *parent_window*, *experiment*, *time_info*, *data_info*, *preproc_info*, *pl_algo_info*, *eval_metrics*, *shuffle_info*, *fs_info=None*, *fs_algo_info=None*, *fs_eval_info=None*, *pl_eval_info=None*, *fold_metrics=None*)

Bases: tkinter.Toplevel

GUI window displaying the results of an experiment.

The window widget extends the class *tkinter.Toplevel*.

Initializes the window widget with all of the information about and results obtained from the given experiment.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this window widget.
- **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.
- **experiment** (*pyplt.experiment.Experiment*) – the given experiment.

- **time_info** (*list of float (size 3)*) – a list containing meta-data about the experiment related to time (the start timestamp (UTC), the end timestamp (UTC), and the duration).
- **data_info** (*list (size 3)*) – a list containing the number of objects, the number of ranks, and the list of data file paths.
- **preproc_info** (*list of dict (size 2)*) – a list containing the include settings dict and the normalization settings dict.
- **pl_algo_info** (*list (size 2)*) – a list containing the algorithm type (*pyplt.util.enums.PLAlgo*) and the string representation of its parameters.
- **eval_metrics** (*dict*) – the evaluation/training results in the form of a dict with keys:
  - '*Training Accuracy*'
  - '*Test Accuracy*' (if applicable)
- **shuffle_info** (*list (size 2)*) – list containing the chosen settings related to shuffling the dataset:
  - shuffle – bool specifying whether or not the dataset was shuffled at the start of the experiment execution.
  - random_seed – optional seed (int or None) used to shuffle the dataset.
- **fs_info** (*list (size 3) or None, optional*) – a list containing the chosen feature selection method type (*pyplt.util.enums.FSMethod*), the string representation of its parameters, and the list of features selected by the feature selection method.
- **fs_algo_info** (*list (size 2) or None, optional*) – a list containing the chosen algorithm type (*pyplt.util.enums.PLAlgo*) for the feature selection stage and the string representation of its parameters.
- **fs_eval_info** (*list (size 2) or None, optional*) – a list containing the evaluation method type (*pyplt.util.enums.EvaluatorType*) for the feature selection stage and the string representation of its parameters.
- **pl_eval_info** (*list (size 2) or None, optional*) – a list containing the evaluation method type (*pyplt.util.enums.EvaluatorType*) and the string representation of its parameters.
- **fold_metrics** (*list of tuple, optional*) – optional fold-specific information (default None) in the form of list of tuples, each containing the start timestamp, end timestamp, evaluation metrics, and a *pandas.DataFrame* representation of the trained model as follows:
  - start_time – *datetime* timestamp (UTC timezone)
  - end_time – *datetime* timestamp (UTC timezone)
  - eval_metrics – dict with keys:
    - '*Training Accuracy*'
    - '*Test Accuracy*' (if applicable)
  - model – *pandas.DataFrame*

Module contents

This package contains GUI-based modules that manage the display of results of an experiment.

Submodules

pyplt.gui.experiment.progresswindow module

class pyplt.gui.experiment.progresswindow.**ProgressWindow**(*parent_window*, *q*, *exec_stopper*)

Bases: `tkinter.Toplevel`

Window widget displaying the execution progress of an experiment via a progress bar and progress log.

Extends the class *tkinter.Toplevel*.

Initializes the ProgressWindow widget object.

Parameters

- **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.
- **q** (*queue.Queue*) – the queue to be used for communication between the thread (*threading.Thread*) carrying out the execution of the experiment and the progress log (*tkinter.Listbox*) of this class.
- **exec_stopper** (*pyplt.util.AbortFlag*) – an abort flag object used to abort the execution before completion if so instructed by the user.

done(*experiment*, *time_info*, *data_info*, *preproc_info*, *fs_info*, *fs_algo_info*, *fs_eval_info*, *pl_algo_info*, *pl_eval_info*, *eval_metrics*, *fold_metrics*, *shuffle_info*)

Update the *ProgressWindow* widget on completion of experiment execution.

Add a 'Success' label, adds a 'Generate Report' button (to open or re-open the results window), and converts the 'Abort' button into a 'Close' button. The window (*pyplt.gui.experiment.results.resultsscreen.ResultsWindow*) containing the experiment details and results is opened automatically.

Parameters

- **experiment** (*pyplt.experiment.Experiment*) – the experiment to be passed on to the results window.
- **time_info** (*list of float (size 3)*) – the time meta-data to be passed on to the results window in the form of a list containing the start timestamp (UTC), end timestamp (UTC), and duration of the experiment.
- **data_info** (*list (size 3)*) – the data to be passed on to the results window in the form of a list containing the number of objects, the number of ranks, and the list of data file paths.
- **preproc_info** (*list of dict (size 2)*) – the pre-processing information to be passed on to the results window in the form of a list containing the include settings dict and the normalization settings dict.
- **fs_info** (*list (size 3)*) – feature selection method information to be passed on to the results window (if applicable) in the form of a list containing the chosen feature selection method type (*pyplt.util.enums.FSMethod*), the string representation of its parameters, and the list of features selected by the feature selection method.
- **fs_algo_info** (*list (size 2)*) – the feature selection algorithm information to be passed on to the results window (if applicable) in the form of a list containing the chosen algorithm type (*pyplt.util.enums.PLAlgo*) and the string representation of its parameters.

- **fs_eval_info** (*list (size 2)*) – the feature selection evaluation method information to be passed on to the results window (if applicable) in the form of a list containing the evaluation method type (*pyplt.util.enums.EvaluatorType*) and the string representation of its parameters.
- **pl_algo_info** (*list (size 2)*) – the preference learning algorithm information to be passed on to the results window in the form of a list containing the algorithm type (*pyplt.util.enums.PLAlgo*) and the string representation of its parameters.
- **pl_eval_info** (*list (size 2)*) – the evaluation method information to be passed on to the results window (if applicable) in the form of a list containing the evaluation method type (*pyplt.util.enums.EvaluatorType*) and the string representation of its parameters.
- **eval_metrics** (*dict*) – the evaluation/training results to be passed on to the results window in the form of a dict with keys:
  - '*Training Accuracy*'
  - '*Test Accuracy*' (if applicable)
- **fold_metrics** (*list of tuple, optional*) – optional fold-specific information (default None) in the form of list of tuples, each containing the start timestamp, end timestamp, evaluation metrics, and a *pandas.DataFrame* representation of the trained model as follows:
  - start_time – *datetime* timestamp (UTC timezone)
  - end_time – *datetime* timestamp (UTC timezone)
  - eval_metrics – dict with keys:
    - '*Training Accuracy*'
    - '*Test Accuracy*' (if applicable)
  - model – *pandas.DataFrame*
- **shuffle_info** (*list (size 2)*) – list containing the chosen settings related to shuffling the dataset:
  - shuffle – bool specifying whether or not the dataset was shuffled at the start of the experiment execution.
  - random_seed – optional seed (int or None) used to shuffle the dataset.

log(*event*)
Display the given string at the end of the progress log (*tkinter.Listbox* object).

Parameters **event** (*str*) – the string to be displayed in the progress log.

progress()
Increment the progress bar (*ttk.Progressbar* object).

put(*item*)
Add a given string item to the queue to be in turn displayed in the progress log.

Parameters **item** (*str*) – a string to be added to the queue and therefore displayed in the progress log.

set_exec_thread(*thread*)
Set the execution thread variable.

update_gui()
Update the GUI.

Hack-y method called while *pyplt.plalgorithms.ranksvm.RankSVM* precomputes kernels from the experiment execution thread to keep the GUI (main) thread going.

The method also sets the flag variable `self._wait` to True immediately before calling `self.update_idletasks()` and sets it back to False immediately after. This is done in order to avoid a deadlock between threads when aborting experiment execution.

pyplt.gui.experiment.singleexperimentwindow module

class pyplt.gui.experiment.singleexperimentwindow.**SingleExperimentWindow**(*parent*)
Bases: `tkinter.Toplevel`

GUI window for setting up and running a single experiment.

Extends the class *tkinter.Toplevel*. Each experiment stage is encapsulated in a separate tab, together comprising a *ttk.Notebook* widget which is controlled by this class. This class also manages the state of the tabs (locked/unlocked) as well as the flow of data between the tabs.

Initializes the *ttk.Notebook* widget in the given window.

The *ttk.Notebook* widget is populated with tabs defined by the classes *pyplt.gui.experiment.dataset.loading.DataLoadingTab*, *pyplt.gui.experiment.preprocessing.preproctab.PreProcessingTab*, *pyplt.gui.experiment.featureselection.featselectiontab.FeatureSelectionTab*, and *pyplt.gui.experiment.preflearning.pltab.PLTab*.

Parameters **parent** (*tkinter.Toplevel*) – the parent window on which this experiment setup window will be stacked.

Module contents

The subpackages and submodules of this package manage the various GUI stages of experiment setup and execution.

pyplt.gui.util package

Submodules

pyplt.gui.util.colours module

This module defines the hexadecimal codes of various colours commonly used throughout the GUI of PLT.

pyplt.gui.util.help module

This module contains a number of classes specifying the content for each of the help/about dialogs throughout PLT.

class pyplt.gui.util.help.**AboutBox**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

'About' window containing text on the details of the PLT software and its license.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the *pyplt.gui.mainmenu.MainMenu* window.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**BeginnerStep1HelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in Step 1 of the BeginnerMenu.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for Step 1 of the BeginnerMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**BeginnerStep2HelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in Step 2 of the BeginnerMenu.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for Step 2 of the BeginnerMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**BeginnerStep3HelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in Step 3 of the BeginnerMenu.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for Step 3 of the BeginnerMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**BeginnerStep4HelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in Step 4 of the BeginnerMenu.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for Step 4 of the BeginnerMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**BeginnerStep5HelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in Step 5 of the BeginnerMenu.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for Step 5 of the BeginnerMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**FSHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the *Feature Selection* tab.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the *Feature Selection* tab.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**HelpDialog**(*parent_window*)
Bases: tkinter.Toplevel

Base class for help dialog windows used to assist the user throughout the GUI.

Initializes the window widget.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**LoadDataHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the *Load Data* tab.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the *Load Data* tab.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**MainHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the *pyplt.gui.mainmenu.MainMenu* window.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the MainMenu.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**PLHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the *Preference Learning* tab.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the *Preference Learning* tab.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**PreprocHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the *Preprocessing* tab.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the *Preprocessing* tab.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**RankDerivationHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user with the rank derivation methods when loading the dataset.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the LoadingParamsWindow.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

class pyplt.gui.util.help.**ResultsHelpDialog**(*parent_window*)
Bases: *pyplt.gui.util.help.HelpDialog*

Help dialog window to assist the user in the experiment report window.

Extends *pyplt.gui.util.help.HelpDialog*.

Initializes the window widget with the help text for the experiment report window.

Parameters **parent_window** (*tkinter.Toplevel*) – the window which this window widget will be stacked on top of.

pyplt.gui.util.styles module

This module defines the various styles used for *ttk* widgets throughout PLT.

pyplt.gui.util.styles.**configure_styles**(*themed_style*)
Define the styles used for ttk widgets throughout the GUI.

pyplt.gui.util.supported_methods module

This module specifies which algorithms and methods are supported by the GUI of PLT.

This module also provides generic helper functions to create instances of algorithm/method classes given enumerated constants representing them.

Developers who would like to add new feature selection methods, preference learning algorithms, and evaluation methods to the GUI of PLT, can do so easily by adding the algorithm/method class and GUI menu class (if applicable) to the dicts declared in this module:

- `supported_fs_methods` – for feature selection methods (in Expert Mode);
- `supported_algorithms` – for preference learning algorithms (in Expert Mode);
- `supported_algorithms_beginner` – for preference learning algorithms (in Beginner Mode);
- `supported_evaluation_methods` – for evaluation methods (in Expert Mode).

pyplt.gui.util.supported_methods.**get_algorithm_instance**(*algorithm_enum*, *params=None*, *beginner_mode=False*)
Create an instance of the preference learning algorithm class represented by the given enum constant.

Each enumerated constant of type *pyplt.util.enums.PLAlgo* in the `supported_algorithms` dict corresponds to a class of type (extending) *pyplt.plalgorithms.base.PLAlgorithm*.

If *params* is specified, the instance is initialized with the given algorithm parameter values. Otherwise, the default values are used.

Parameters

- **algorithm_enum** (*pyplt.util.enums.PLAlgo*) – the algorithm type (enum).
- **params** (*dict or None, optional*) – optional algorithm parameter values in the form of a dict (default None). The keys of the dict should match the keywords of the arguments that would be passed to the corresponding *pyplt.plalgorithms.base.PLAlgorithm* constructor. For example, for the *Backpropagation* algorithm the dict should contain the following items:
  - ann_topology: the topology of the neurons in the network
  - learn_rate: the learning rate
  - error_threshold: the error threshold
  - epochs: the number of epochs
  - activation_functions: the activation functions for each neuron layer in the network

  On the other hand, for the *RankSVM* algorithm the dict should contain the following items:
  - kernel: the kernel name
  - gamma: the gamma kernel parameter value
  - degree: the degree kernel parameter value
- **beginner_mode** (*bool, optional*) – specifies whether or not the algorithm is being used in the beginner mode (default False).

Returns an instance of the class corresponding to the given algorithm.

Return type *pyplt.plalgorithms.base.PLAlgorithm*

Raises ***InvalidParameterValueException*** – if the user attempted to use a value smaller or equal to 0.0 for the *gamma* parameter of the *RankSVM* algorithm.

pyplt.gui.util.supported_methods.**get_eval_method_instance**(*eval_method_enum*, *params=None*)

Create an instance of the evaluation method class represented by the given enum constant.

Each enumerated constant of type *pyplt.util.enums.EvaluatorType* in the supported_evaluation_methods dict corresponds to a class of type (extending) *pyplt.evaluation.base.Evaluator*.

If *params* is specified, the instance is initialized with the given evaluation method parameter values. Otherwise, the default values are used.

Parameters

- **eval_method_enum** (*pyplt.util.enums.EvaluatorType*) – the evaluation method type (enum).
- **params** (*dict or None, optional*) – optional evaluation method parameter values in the form of a dict (default None). The keys of the dict should match the keywords of the arguments that would be passed to the corresponding *pyplt.evaluation.base.Evaluator* constructor. For example, for the *Holdout* method, the dict should contain the following items:
  - test_proportion: a float specifying the proportion of data to be used as training data (the rest is to be used as test data) or None

  On the other hand, for the *KFoldCrossValidation* method, the dict should contain the following items:

- k: the number of folds to uniformly split the data into when using the automatic approach or None
- test_folds: an array specifying the fold index for each sample in the dataset when using the manual approach or None

Returns an instance of the class corresponding to the given evaluation method.

Return type *pyplt.evaluation.base.Evaluator*

Raises

- ***InvalidParameterValueException*** – if the user attempts to use a value smaller than 2 for the *k* parameter of K-Fold Cross Validation.
- ***MissingManualFoldsException*** – if the user chooses to specify folds manually for cross validation but fails to load the required file containing the fold IDs.

pyplt.gui.util.supported_methods.**get_fs_method_instance**(*fs_method_enum*, *params=None*)

Create an instance of the feature selection method class represented by the given enum constant.

Each enumerated constant of type *pyplt.util.enums.FSMethod* in the supported_fs_methods dict corresponds to a class of type (extending) *pyplt.fsmethods.base.FeatureSelectionMethod*.

If *params* is specified, the instance is initialized with the given feature selection method parameter values. Otherwise, the default values are used.

Parameters

- **fs_method_enum** (*pyplt.util.enums.FSMethod*) – the feature selection method type (enum).
- **params** (*dict or None, optional*) – optional feature selection method parameter values in the form of a dict (default None). The keys of the dict should match the keywords of the arguments that would be passed to the corresponding *pyplt.fsmethods.base.FeatureSelectionMethod* constructor.

Returns an instance of the class corresponding to the given feature selection method.

Return type *pyplt.fsmethods.base.FeatureSelectionMethod*

pyplt.gui.util.tab_locking module

This module defines two classes which manage the locking and unlocking of tabs in *ttk.Notebook* widgets.

class pyplt.gui.util.tab_locking.**LockableTab**(*parent*, *parent_window*)

Bases: tkinter.Frame

Base class for creating *ttk.Notebook* tabs that are easily locked and unlocked.

Extends the class *tkinter.Frame*.

Initializes the tab.

The locking and unlocking mechanism works by switching between which child frame of the base frame self._base is raised. The 'locked' frame of standard type LockedFrame is raised when the tab is to be locked whereas the 'unlocked' frame is raised when the tab is to be unlocked. The 'unlocked' frame is specified by the user by implementing the abstract method get_normal_frame(). The method extends the __init__() method in tk.Frame.

Parameters

- **parent** (*tkinter widget*) – the parent widget of this LockableTab widget.

- **parent_window** (*tkinter.Toplevel*) – the window which will contain this LockableTab widget.

get_base_frame()
Return the base frame widget.

get_normal_frame()
Abstract method to be implemented in subclasses to return the 'unlocked' frame.

lock()
Raise the 'locked' child frame of the base frame over the 'unlocked' child frame of the base frame.

unlock()
Raise the 'unlocked' child frame of the base frame over the 'locked' child frame of the base frame.

class pyplt.gui.util.tab_locking.**LockedFrame**(*parent*)
Bases: tkinter.Frame

Standard class to be used as the 'locked' frame in a LockableTab object.

Extends the class *tkinter.Frame*.

Populates the frame with a simple label indicating the tab's status as 'locked'.

Parameters **parent** (*tkinter widget*) – the parent widget of this LockableFrame widget.

pyplt.gui.util.text module

This module contains a number of helper functions for displaying text properly in the GUI.

pyplt.gui.util.text.**real_param_name**(*param_name*)
Receive a parameter name and return the grammatically-correct version of its name.

Parameters **param_name** (*str*) – the parameter name as used internally by PLT.

Returns the grammatically-correct version of param_name.

Return type str

pyplt.gui.util.text.**real_type_name**(*type_name*)
Receive a method type name and return the grammatically-correct version of its name.

Parameters **type_name** (*str*) – the method type name as used internally by PLT.

Returns the grammatically-correct version of type_name.

Return type str

pyplt.gui.util.windowstacking module

class pyplt.gui.util.windowstacking.**Mode**
Bases: enum.Enum

Class specifying enumerated constants for the different modes of window stacking.

Extends *enum.Enum*.

OPEN_ONLY = 1

WITH_CLOSE = 0

pyplt.gui.util.windowstacking.**disable_parent**(*self*, *parent*, *mode*)
Change state of parent window to 'disabled'.

Parameters

- **self** (*tkinter.Toplevel*) – the new window being stacked on top.
- **parent** (*tkinter.Toplevel*) – the parent of the new window widget to be stacked above it.
- **mode** (*pyplt.gui.util.windowstacking.Mode*) – the window stacking mode.

pyplt.gui.util.windowstacking.**on_close**(*new_window*, *parent*)

Restore the parent window and all its children widgets to their 'normal' state and bring it back on top.

The parent window is elevated back to the topmost level of the stack of windows.

This method is toggled when a stacked window is closed and its stacking mode was set to WITH_CLOSE.

Parameters

- **new_window** (*tkinter.Toplevel*) – the stacked window being closed.
- **parent** (*tkinter.Toplevel*) – the parent window of new_window.

pyplt.gui.util.windowstacking.**place_window**(*self*, *parent*, *position=1*)

Set the geometry of child window to a given position relative to the parent window's geometry.

Parameters

- **self** (*tkinter.Toplevel*) – the new window being stacked on top.
- **parent** (*tkinter.Toplevel*) – the parent of the new window widget to be stacked above it.
- **position** (*int: 0 or 1, optional*) – the position at which to place the new window relative to the parent window's geometry. If 0, the new window will be placed directly on top of the parent window. If 1, the new window will be placed to the side of the parent window (default 1).

pyplt.gui.util.windowstacking.**stack_window**(*new_win*, *parent_win*)

Elevate the new window to the topmost level of the stack of windows.

Parameters

- **new_win** (*tkinter.Toplevel*) – the new window to be stacked on top.
- **parent_win** (*tkinter.Toplevel*) – the parent window of new_win.

Module contents

This package defines a number of utility classes for the GUI of PLT.

Submodules

pyplt.gui.beginnermenu module

class pyplt.gui.beginnermenu.**BeginnerMenu**(*parent*)

Bases: tkinter.Toplevel

GUI window containing a simplified PLT experiment set-up menu for beginners.

Extends the class *tkinter.Toplevel*.

Initializes the *BeginnerMenu* object.

Parameters **parent** (*tkinter.Toplevel*) – the parent window on top of which this *tkinter.Toplevel* window will be stacked.

pyplt.gui.mainmenu module

class pyplt.gui.mainmenu.**MainMenu**(*master*)
Bases: object

Main menu window widget.

This menu allows the user to select whether to run PLT in a Beginner mode or Advanced mode.

Initializes and populates the main menu window widget.

Parameters **master** (*tkinter.Tk*) – the root window on which the main menu will be displayed.

Module contents

This package contains several modules that manage the Graphical User Interface (GUI) component of PLT.

pyplt.plalgorithms package

Submodules

pyplt.plalgorithms.backprop_tf module

class pyplt.plalgorithms.backprop_tf.**BackpropagationTF**(*ann_topology=None*, *learn_rate=0.001*, *error_threshold=0.001*, *epochs=10*, *activation_functions=None*, *batch_size=32*, *debug=False*)
Bases: *pyplt.plalgorithms.base.PLAlgorithm*

Backpropagation algorithm implemented with the *tensorflow* package.

This is a gradient-descent algorithm that iteratively (over a given number of epochs) optimizes an error function by adjusting the weights of an artificial neural network (ANN) model proportionally to the gradient of the error with respect to the current value of the weights and current data samples. The proportion and therefore the strength of each update is regulated by the given learning rate. The error function used is the Rank Margin function which for a given pair of data samples (A and B, with A preferred over B) yields 0 if the network output for A (fA) is more than one unit larger than the network output for B (fB) and 1.0-((fA)-(fB)) otherwise. The total error is averaged over the complete set of pairs in the training set. If the error is below a given threshold, training stops before reaching the specified number of epochs, and the current weight values are returned as the final model. In PLT, the algorithm was implemented using the *tensorflow* library.

Initializes the BackpropagationTF object.

Parameters

- **ann_topology** (*list or None, optional*) – a list indicating the topology of the artificial neural network (ANN) to be used with the algorithm. The list contains the number of neurons in each layer of the ANN, excludes the input layer but including the output layer (must always be 1 neuron in size); a value of None is equivalent to [1] indicating an ANN with no hidden layers and only an output layer (consisting of 1 neuron) (default None).

- **learn_rate** (*float, optional*) – the learning rate used in the weight update step of the Backpropagation algorithm (default 0.001).
- **error_threshold** (*float, optional*) – a threshold at or below which the error of a model is considered to be sufficiently trained (default 0.001).
- **epochs** (*int, optional*) – the maximum number of iterations the algorithm should make over the entire pairwise rank training set (default 10).
- **activation_functions** (list of *pyplt.util.enums.ActivationType* or None, optional) – a list of the activation functions to be used across the neurons for each layer of the ANN (default None); if None, all layers will use the Rectified Linear Unit (ReLU) function i.e. *pyplt.util.enums.ActivationType.RELU*, except for the output layer which will use the Logistic Sigmoid function i.e. *pyplt.util.enums.ActivationType.SIGMOID*.
- **batch_size** (*int, optional*) – number of samples per gradient update (default 32).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).

calc_train_accuracy(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

An algorithm-specific approach to calculating the training accuracy of the learned model.

This method is implemented explicitly for this algorithm since this approach is substantially more efficient for algorithms using the *tensorflow* package than the calc_train_accuracy() method of *pyplt.plalgorithms.base.PLAlgorithm* objects allows.

The training accuracy is determined by calculating the percentage of how many of the training ranks the model is able to predict correctly.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data the model was trained on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data the model was trained on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the training accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

clean_up()

Close the *tensorflow* session once the algorithm class instance is no longer needed.

V.IMP. THIS FUNCTION MUST BE CALLED WHEN THE CLASS INSTANCE IS NO LONGER IN USE unless a context manager is used around the BackpropagationTF class instance!!!

init_train(*n_features*)
Initialize the model (topology).

This method is to be called if one wishes to initialize the model (topology) explicitly. This is done by declaring *tensorflow* placeholders, variables, and operations. This may be used, for example, to use the same *BackpropagationTF* object but simply modify its topology while evaluating different feature sets during wrapper-type feature selection processes. If not called explicitly, the *train()* method will call it once implicitly.

Parameters **n_features** (*int*) – the number of features to be used during the training process.

load_model()
Load a model which was trained using this algorithm. # TODO: to be actually implemented.

predict(*input_object*, *progress_window=None*, *exec_stopper=None*)
Predict the output of a given input object by running it through the learned model.

Parameters

- **input_object** (one row from a *pandas.DataFrame*) – the input data corresponding to a single object.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns the predicted output resulting from running the learned model using the given input.

Return type float

save_model(*timestamp*, *path=''*, *suppress=False*)
Save the ANN model to a Comma Separated Value (CSV) file at the path indicated by the user.

Optionally, the file creation may be suppressed and a *pandas.DataFrame* representation of the model returned instead.

The file/DataFrame stores the weights, biases, and activation functions of each neuron in each layer of the ANN. Each row represents these values for a neuron in a layer, starting from the first neuron in the first hidden layer (if applicable), and moving forward neuron-by-neuron, layer-by-layer, until the output neuron is reached. The number of columns is variable as the file stores enough columns to represent the maximum number of weights across all neurons in the network.

Weights columns are labeled with the letter 'w' followed by the index of the incoming neuron from which the given weight is connected the current neuron. Hidden layers in the 'layer' column are labelled with the letter 'h' followed by the index of the layer. The output layer is simply labelled as 'OUTPUT'.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default ""). If "", the file is saved to a logs folder in the project root directory by default.

- **suppress** (*bool, optional*) – specifies whether or not to suppress the file creation and return a *pandas.DataFrame* representation of the model instead (default False).

Returns a *pandas.DataFrame* representation of the model, if the *suppress* parameter was set to True, otherwise None.

Return type

- *pandas.DataFrame* – if *suppress* is True
- None – otherwise

test (*objects*, *test_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
An algorithm-specific approach to testing/validating the model using the given test data.

This method is implemented explicitly for this algorithm since this approach is substantially more efficient for algorithms using the *tensorflow* package than the test() method of the base class *pyplt.plalgorithms.base.PLAlgorithm*.

Parameters

- **objects** (*pandas.DataFrame*) – the objects data for the model to be tested/validated on.
- **test_ranks** (*pandas.DataFrame*) – the pairwise rank data for the model to be tested/validated on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used during the testing/validation process; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the test/validation accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

train (*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
Run a *tensorflow* session to infer an ANN model using the given training data.

The given pairwise rank data is split into a set of preferred objects and a set of non-preferred objects, which are then fed into the ANN. The resulting (predicted) model output of each object in a given rank pair is compared to the actual preference and the error is calculated via a Rank Margin error function. The algorithm attempts to optimize the average error over the entire set of ranks across several iterations (epochs) until it reaches the maximum number number of iterations (epochs) or reaches the error threshold.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data to train the model on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data to train the model on.

- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- True – if execution is completed successfully.
- None – if experiment is aborted before completion by *exec_stopper*.

static transform_data(*object_*)

Transform an object into the format required by this particular algorithm implementation.

In this case, nothing changes.

Parameters **object** (one row from a *pandas.DataFrame*) – the object to be transformed.

Returns the transformed object in the form of an array.

Return type *numpy.ndarray*

pyplt.plalgorithms.base module

class pyplt.plalgorithms.base.**PLAlgorithm**(*description='A preference learning algorithm.', name='', debug=False, **kwargs*)

Bases: object

Base class for all preference learning algorithms.

Initializes the PLAlgorithm object.

Parameters

- **description** (*str, optional*) – a description of the algorithm (default "A preference learning algorithm.").
- **name** (*str, optional*) – the name of the algorithm (default "").
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).
- **kwargs** – any additional parameters for the algorithm.

calc_train_accuracy(*train_objects, train_ranks, use_feats=None, progress_window=None, exec_stopper=None*)

Base method for calculating the training accuracy of the learned model.

The training accuracy is determined by calculating the percentage of how many of the training ranks the model is able to predict correctly.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data the model was trained on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data the model was trained on.

- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the training accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

clean_up()
Base method for any potential final clean up instructions to be carried out.

Does nothing unless overriden in child class.

get_description()
Get the preference learning algorithm.

Returns the description of the algorithm.

Return type str

get_name()
Get the name of the preference learning algorithm.

Returns the name of the algorithm.

Return type str

get_params()
Return all additional parameters of the preference learning algorithm (if applicable).

Returns a dict containing all additional parameters of the algorithm with the parameter names as the dict's keys and the corresponding parameter values as the dict's values (if applicable).

Return type dict

get_params_string()
Return a string representation of all additional parameters of the preference learning algorithm (if applicable).

Returns the string representation of all additional parameters of the algorithm (if applicable).

Return type str

get_train_accuracy()
Get the training accuracy of the learned model.

Returns the training accuracy of the learned model.

Return type float

init_train(*n_features*)
Abstract method for carrying out any initializations prior to the training stage.

All children classes must implement this method.

Parameters **n_features** (*int*) – the number of features to be used during the training process.

load_model()
Abstract method for loading a model which was trained using this algorithm.

All children classes must implement this method.

predict(*input_object*, *progress_window=None*, *exec_stopper=None*)
Abstract method for predicting the output of a given input by running it through the learned model.

All children classes must implement this method.

Parameters

- **input_object** (one row from a *pandas.DataFrame*) – the input data corresponding to a single object.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns a list containing the predicted output resulting from running the learned model using the given input.

Return type list of float (size 1)

save_model(*timestamp*, *path=''*, *suppress=False*)
Abstract model to save the model to file.

Optionally, the file creation may be suppressed and a *pandas.DataFrame* representation of the model returned instead.

All children classes must implement this method.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default ""). If "", the file is saved to a logs folder in the project root directory by default.
- **suppress** (*bool, optional*) – specifies whether or not to suppress the file creation and return a *pandas.DataFrame* representation of the model instead (default False).

Returns a *pandas.DataFrame* representation of the model, if the *suppress* parameter was set to True, otherwise None.

Return type

- *pandas.DataFrame* – if *suppress* is True
- None – otherwise

save_model_with_dialog(*timestamp*, *parent_window*, *suffix=''*)
Open a file dialog window (GUI) and save the learned model to file at the path indicated by the user.

The model file must be a Comma Separated Value (CSV)-type file with the extension '.csv'.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **parent_window** (*tkinter.Toplevel*) – the window widget which the file dialog window widget will be stacked on top of.
- **suffix** (*str, optional*) – an additional string to add at the end of the file name (default "").

Returns specifies whether or not the file was successfully saved.

Return type bool

test(*objects*, *test_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

Base method for calculating the prediction accuracy of the learned model on a given dataset (test set).

The prediction accuracy is determined by calculating the percentage of how many of the test ranks the model is able to predict correctly.

Parameters

- **objects** (*pandas.DataFrame*) – the objects data to be predicted by the model.
- **test_ranks** (*pandas.DataFrame*) – the pairwise rank data to be predicted by the model.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used during the prediction process; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the prediction accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

train(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

Abstract method for the training stage in the machine learning process.

All children classes must implement this method.

Parameters

- **train_objects** (*pandas.DataFrame*) – containing the objects data to train the model on.
- **train_ranks** (*pandas.DataFrame*) – containing the pairwise rank data to train the model on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel*

widget) used to display a progress log and progress bar during the experiment execution (default None).

- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- True or any other value – if execution is completed successfully.
- None – if experiment is aborted before completion by *exec_stopper*.

static transform_data(*object_*)

Abstract method to transform a sample (object) into the format required by this particular algorithm implementation.

All children classes must implement this method.

Parameters **object** (one row from a *pandas.DataFrame*) – the data sample (object) to be transformed.

Returns the transformed object.

pyplt.plalgorithms.ranknet module

class pyplt.plalgorithms.ranknet.**RankNet**(*ann_topology=None*, *learn_rate=0.001*, *epochs=100*, *hidden_activation_functions=None*, *batch_size=32*, *debug=False*)

Bases: *pyplt.plalgorithms.base.PLAlgorithm*

RankNet algorithm implemented with the *keras* package.

The RankNet algorithm is an extension of the Backpropagation algorithm which uses a probabilistic cost function to handle ordered pairs of data. As in Backpropagation, the algorithm iteratively (over a given number of epochs) optimizes the error function by adjusting the weights of an artificial neural network (ANN) model proportionally to the gradient of the error with respect to the current value of the weights and current data samples. The error function used is the binary cross-entropy function. The proportion and therefore the strength of each update is regulated by the given learning rate. The total error is averaged over the complete set of pairs in the training set. In PLT, the algorithm was implemented using the *keras* library.

Initialize the RankNet instance.

Parameters

- **ann_topology** (*list or None, optional*) – a list indicating the topology of the artificial neural network (ANN) to be used with the algorithm. The list contains the number of neurons in each layer of the ANN, excludes the input layer but including the output layer (must always be 1 neuron in size); a value of None is equivalent to [1] indicating an ANN with no hidden layers and only an output layer (consisting of 1 neuron) (default None).
- **hidden_activation_functions** (list of pyplt.plalgorithms.backprop_tf.ActivationType or None, optional) – a list of the activation function to be used across the neurons for each hidden layer of the ANN; if None, all hidden layers will use the Rectified Linear Unit (ReLU) function i.e. pyplt.plalgorithms.backprop_tf.ActivationType.RELU (default None). Note that this parameter excludes the activation function at the output layer of the network which is fixed.
- **learn_rate** (*float, optional*) – the learning rate used in the weight update step of the Backpropagation algorithm (default 0.001).

- **epochs** (*int, optional*) – the maximum number of iterations the algorithm should make over the entire pairwise rank training set (default 10).
- **batch_size** (*int, optional*) – number of samples per gradient update (default 32).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).

calc_train_accuracy(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

An algorithm-specific approach to calculating the training accuracy of the learned model.

This method is implemented explicitly for this algorithm since this approach is substantially more efficient for algorithms using the *keras* package than the calc_train_accuracy() method of *pyplt.plalgorithms.base.PLAlgorithm* objects allows.

The training accuracy is determined by calculating the percentage of how many of the training ranks the model is able to predict correctly.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data the model was trained on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data the model was trained on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the training accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

clean_up()

Close the backend *tensorflow* session once the algorithm class instance is no longer needed.

init_train(*n_features*)

Initialize the model (topology).

This is done by declaring *keras* placeholders, variables, and operations. This may also be used, for example, to simply modify (re-initialize) the topology of the model while evaluating different feature sets during wrapper-type feature selection processes.

Parameters **n_features** (*int*) – the number of features to be used during the training process.

predict(*input_object*, *progress_window=None*, *exec_stopper=None*)

Abstract method for predicting the output of a given input by running it through the learned model.

All children classes must implement this method.

Parameters

- **input_object** (one row from a *pandas.DataFrame*) – the input data corresponding to a single object.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns a list containing the predicted output resulting from running the learned model using the given input.

Return type list of float (size 1)

save_model(*timestamp*, *path=''*, *suppress=False*)

Save the trained model to file in a human-readable format.

Optionally, the file creation may be suppressed and a *pandas.DataFrame* representation of the model returned instead.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default ""). If "", the file is saved to a logs folder in the project root directory by default.
- **suppress** (*bool, optional*) – specifies whether or not to suppress the file creation and return a *pandas.DataFrame* representation of the model instead (default False).

Returns a *pandas.DataFrame* representation of the model, if the *suppress* parameter was set to True, otherwise None.

Return type

- *pandas.DataFrame* – if *suppress* is True
- None – otherwise

test(*objects*, *test_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

An algorithm-specific approach to testing/validating the model using the given test data.

This method is implemented explicitly for this algorithm since this approach is substantially more efficient for algorithms using the *keras* package than the test() method of the base class *pyplt.plalgorithms.base.PLAlgorithm*.

Parameters

- **objects** (*pandas.DataFrame*) – the objects data for the model to be tested/validated on.
- **test_ranks** (*pandas.DataFrame*) – the pairwise rank data for the model to be tested/validated on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used during the testing/validation process; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel*

widget) used to display a progress log and progress bar during the experiment execution (default None).

- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the test/validation accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

train(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

Infer an ANN model using the given training data.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data to train the model on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data to train the model on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- True – if execution is completed successfully.
- None – if experiment is aborted before completion by *exec_stopper*.

transform_data(*object_*)

Transform a sample (object) into the format required by this particular algorithm implementation.

In this case, no transformation is needed.

Parameters **object** (one row from a *pandas.DataFrame*) – the data sample (object) to be transformed.

Returns the transformed object (same as **object_** in this case).

pyplt.plalgorithms.ranksvc module

class pyplt.plalgorithms.ranksvc.**RankSVC**(*kernel=<KernelType.RBF: 1>*, *gamma='auto'*, *degree=3*, *debug=False*)

Bases: *pyplt.plalgorithms.base.PLAlgorithm*

RankSVM algorithm implemented using the *scikit-learn* library.

N.B. This implementation is similar to the implementation in the *pyplt.plalgorithms.ranksvm.RankSVM* class but instead of using the *OneClassSVM* class of the *scikit-learn* libary, this implementation uses the *SVC* class of the same library. The input and output of the model are treated differently as the SVC model is a binary classifier (see *pairwise_transform_from_ranks()*). Consequently, unlike the RankSVM

implementation, the model cannot predict a real-valued output for a single object/instance. Rather, the model can only be used on pairs of objects in order for the output to make sense. This implementation is only available in the API of PLT.

A Support Vector Machine (SVM) is a binary classifier that separates the input put samples linearly in a projected space. The decision boundary of the classifier is given by a linear combination of training samples (called support vectors) in the projected space. The projection in provided by the kernel function that the user must select. The support vector and weights are selected to satisfy a set of constrains derived from the input samples and a cost parameter which regulates the penalization of misclassified training samples. In PLT, the algorithm was implemented using the *scikit-learn* library. In this implementation, the quadratic programmer solver contained in LIBSVM is used. The RankSVM algorithm is a rank-based version of traditional SVM training algorithms. It uses the same solver as standard training algorithms for binary SVMs; the only difference lies in the set of constraints which are defined in terms of pairwise preferences between training samples.

Initializes the RankSVM object.

Parameters

- **kernel** (*pyplt.util.enums.KernelType*, optional) – the kernel function mapping the input samples to the projected space (default *pyplt.util.enums.KernelType.RBF*).
- **gamma** (*float or 'auto', optional*) – the kernel coefficient for the 'rbf', 'poly' and 'sigmoid' kernels. If gamma is set to 'auto' then 1/n_features will be used instead (default 'auto').
- **degree** (*float, optional*) – the degree of the polynomial ('poly') kernel function (default 3).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).

Raises ***InvalidParameterValueException*** – if the user attempts to use a gamma value <= 0.0.

calc_train_accuracy(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

An algorithm-specific approach to calculates the training accuracy of the learned model.

This method is tailored specifically for this algorithm implementation and therefore replaces the calc_train_accuracy() method of *pyplt.plalgorithms.base.PLAlgorithm*.

The training accuracy is determined by calculating the percentage of how many of the training ranks the binary classification model is able to predict correctly.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data the model was trained on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data the model was trained on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the training accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

pairwise_transform_from_ranks(*objects*, *ranks*, *use_feats=None*)

Convert a rank-based dataset into the required format for use by RankSVM prior to the training stage.

For each rank (pair of objects) in *ranks*, a feature vector subtraction is carried out between the two objects (both feature vectors) from either side (i.e., a-b and b-a for a given pair of objects/feature vectors a and b where a is preferred over b) and stored as a new transformed data point in *X_trans*. Additionally, for each positive difference (a-b), a value of +1 is stored as its corresponding target class label in *y_trans* whereas value of -1 is stored for each negative difference (b-a).

Parameters

- **objects** (*pandas.DataFrame*) – the objects data to be converted.
- **ranks** (*pandas.DataFrame*) – the pairwise rank data to be converted.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training (default None). If None, all original features are used.

Returns

a tuple containing:

- the converted dataset ready to be used by RankSVM in the form of two arrays:
  - array of shape (n_ranks*2, n_feaures) which stores the positive and negative feature vector differences for each rank.
  - array of shape n_ranks*2 which stores the corresponding target class labels (alternating +1s and -1s).
- a copy of the actual objects data (*pandas.DataFrame*) used in the transformation.

Return type tuple (size 3)

save_model(*timestamp*, *path=''*, *suppress=False*)

Save the RankSVM model to a Comma Separated Value (CSV) file at the path indicated by the user.

Optionally, the file creation may be suppressed and a *pandas.DataFrame* representation of the model returned instead.

The file/DataFrame stores support vectors and corresponding alpha values of the SVM model.

The first column contains the support vectors each representing an object ID. The second column contains the alpha values corresponding to the support vectors in the first column.

The parameters (kernel, gamma and degree) used to construct the model are stored within the file name.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default ''). If '', the file is saved to a logs folder in the project root directory by default. The kernel, gamma, and degree parameters are automatically included in the file name.

- **suppress** (*bool, optional*) – specifies whether or not to suppress the file creation and return a *pandas.DataFrame* representation of the model instead (default False).

Returns a *pandas.DataFrame* representation of the model, if the *suppress* parameter was set to True, otherwise None.

Return type

- *pandas.DataFrame* – if *suppress* is True
- None – otherwise

test (*objects*, *test_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
An algorithm-specific approach to testing/validating the model using the given test data.

Parameters

- **objects** (*pandas.DataFrame*) – the objects data that the model was trained on.
- **test_ranks** (*pandas.DataFrame*) – the pairwise rank data for the model to be tested/validated on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used during the testing/validation process; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the test/validation accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

train (*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
Train a RankSVM model on the given training data.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data to train the model on.
- **train_ranks** (*pandas DataFrame*) – the pairwise rank data to train the model on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns None – if experiment is aborted before completion by *exec_stopper*.

static transform_data(*object_*)

Transform an object into the format required by this particular implementation of RankSVM.

Parameters **object** (one row from a *pandas.DataFrame*) – the object to be transformed.

Returns the transformed object in the form of an array.

Return type *numpy.ndarray*

pyplt.plalgorithms.ranksvm module

class pyplt.plalgorithms.ranksvm.**RankSVM**(*kernel=<KernelType.RBF: 1>*, *gamma='auto'*, *degree=3*, *debug=False*)

Bases: *pyplt.plalgorithms.base.PLAlgorithm*

RankSVM algorithm implemented using the *scikit-learn* library.

A Support Vector Machine (SVM) is a binary classifier that separates the input put samples linearly in a projected space. The decision boundary of the classifier is given by a linear combination of training samples (called support vectors) in the projected space. The projection in provided by the kernel function that the user must select. The support vector and weights are selected to satisfy a set of constrains derived from the input samples and a cost parameter which regulates the penalization of misclassified training samples. In PLT, the algorithm was implemented using the *scikit-learn* library. In this implementation, the quadratic programmer solver contained in LIBSVM is used. The RankSVM algorithm is a rank-based version of traditional SVM training algorithms. It uses the same solver as standard training algorithms for binary SVMs; the only difference lies in the set of constraints which are defined in terms of pairwise preferences between training samples.

Initializes the RankSVM object.

Parameters

- **kernel** (*pyplt.util.enums.KernelType*, optional) – the kernel function mapping the input samples to the projected space (default *pyplt.util.enums.KernelType.RBF*).
- **gamma** (*float or 'auto', optional*) – the kernel coefficient for the 'rbf', 'poly' and 'sigmoid' kernels. If gamma is set to 'auto' then 1/n_features will be used instead (default 'auto').
- **degree** (*float, optional*) – the degree of the polynomial ('poly') kernel function (default 3).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging (default False).

Raises ***InvalidParameterValueException*** – if the user attempts to use a gamma value <= 0.0.

calc_train_accuracy(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)

An algorithm-specific approach to calculates the training accuracy of the learned model.

This method is tailored specifically for this algorithm implementation and therefore replaces the calc_train_accuracy() method of *pyplt.plalgorithms.base.PLAlgorithm*.

The training accuracy is determined by calculating the percentage of how many of the training ranks the model is able to predict correctly.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data the model was trained on.
- **train_ranks** (*pandas.DataFrame*) – the pairwise rank data the model was trained on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the training accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

predict_m(*input_objects*, *progress_window=None*, *exec_stopper=None*)

Predict the output of a given set of input samples by running them through the learned RankSVM model.

Parameters

- **input_objects** (*numpy.ndarray*) – array of shape [n_samples, n_feats] containing the input data corresponding to a set of (test) objects.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- a list containing the average predicted output resulting from running the learned model using the given input objects – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type list of float (size 1)

save_model(*timestamp*, *path=''*, *suppress=False*)

Save the RankSVM model to a Comma Separated Value (CSV) file at the path indicated by the user.

Optionally, the file creation may be suppressed and a *pandas.DataFrame* representation of the model returned instead.

The file/DataFrame stores support vectors and corresponding alpha values of the SVM model.

The first column contains the support vectors each representing a rank in the form of a tuple (int, int) containing the ID of the preferred object in the pair, followed by the ID of the non-preferred object in the pair. The second column contains the alpha values corresponding to the support vectors in the first column.

The parameters (kernel, gamma and degree) used to construct the model are stored within the file name.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default “”). If “”, the file is saved to a logs folder in the project root directory by default. The kernel, gamma, and degree parameters are automatically included in the file name.
- **suppress** (*bool, optional*) – specifies whether or not to suppress the file creation and return a *pandas.DataFrame* representation of the model instead (default False).

Returns a *pandas.DataFrame* representation of the model, if the *suppress* parameter was set to True, otherwise None.

Return type

- *pandas.DataFrame* – if *suppress* is True
- None – otherwise

test(*objects*, *test_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
An algorithm-specific approach to testing/validating the model using the given test data.

Parameters

- **objects** (*pandas.DataFrame*) – the objects data that the model was trained on.
- **test_ranks** (*pandas.DataFrame*) – the pairwise rank data for the model to be tested/validated on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used during the testing/validation process; if None, all original features are used (default None).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the test/validation accuracy of the learned model – if execution is completed successfully.
- None – if aborted before completion by *exec_stopper*.

Return type float

train(*train_objects*, *train_ranks*, *use_feats=None*, *progress_window=None*, *exec_stopper=None*)
Train a RankSVM model on the given training data.

Parameters

- **train_objects** (*pandas.DataFrame*) – the objects data to train the model on.
- **train_ranks** (*pandas DataFrame*) – the pairwise rank data to train the model on.
- **use_feats** (*list of str or None, optional*) – a subset of the original features to be used when training; if None, all original features are used (default None).

- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- True – if execution is completed successfully.
- None – if experiment is aborted before completion by *exec_stopper*.

static transform_data(*object_*)

Transform an object into the format required by this particular implementation of RankSVM.

Parameters **object** (one row from a *pandas.DataFrame*) – the object to be transformed.

Returns the transformed object in the form of an array.

Return type *numpy.ndarray*

Module contents

This package contains backend modules that manage the preference learning step of an experiment.

pyplt.util package

Submodules

pyplt.util.enums module

This module contains a number of classes defining different types of enumerated constants used throughout PLT.

class pyplt.util.enums.**ActivationType**

Bases: enum.Enum

Class specifying enumerated constants for types of activation functions used by Backpropagation.

Extends *enum.Enum*.

LINEAR = 0

RELU = 2

SIGMOID = 1

class pyplt.util.enums.**DataSetType**

Bases: enum.Enum

Class specifying enumerated constants for types of ranks present in data sets.

Extends the class enum.Enum.

ORDERED = 2

PREFERENCES = 1

class pyplt.util.enums.**EvaluatorType**
Bases: enum.Enum

Class specifying enumerated constants for evaluators.

Extends *enum.Enum*.

HOLDOUT = 1

KFCV = 2

NONE = 0

class pyplt.util.enums.**FSMethod**
Bases: enum.Enum

Class specifying enumerated constants for feature selection methods.

Extends *enum.Enum*.

NONE = 0

N_BEST = 1

SBS = 3

SFS = 2

class pyplt.util.enums.**FileType**
Bases: enum.Enum

Class specifying enumerated constants for data file types.

Extends the class enum.Enum.

OBJECTS = 1

RANKS = 2

SINGLE = 3

class pyplt.util.enums.**KernelType**
Bases: enum.Enum

Class specifying enumerated constants for kernels used by RankSVM.

Extends *enum.Enum*.

LINEAR = 0

POLY = 2

RBF = 1

class pyplt.util.enums.**NormalizationType**
Bases: enum.Enum

Class specifying enumerated constants for data normalization methods.

Extends the class enum.Enum.

MIN_MAX = 1

NONE = 0

Z_SCORE = 2

class pyplt.util.enums.**PLAlgo**
Bases: enum.Enum

Class specifying enumerated constants for preference learning algorithms.

Extends *enum.Enum*.

BACKPROPAGATION = 2

BACKPROPAGATION_SKLEARN = 3

NEUROEVOLUTION = 4

RANKNET = 5

RANKSVM = 1

class pyplt.util.enums.**ParamType**
Bases: enum.Enum

Class specifying enumerated constants for parameter types.

Extends *enum.Enum*.

FLOAT = 1

FLOAT_POSITIVE = 2

INT = 0

Module contents

This package defines a number of utility classes for backend processes of PLT.

class pyplt.util.**AbortFlag**
Bases: object

This utility class assists the termination of experiments before completion.

Initializes a stopping flag variable to False (boolean).

The stopping variable indicates whether or not the experiment should be stopped.

stop()
Set the stopping flag to True.

stopped()
Get the stopping flag which indicates whether or not the experiment should be stopped.

Returns the stopping flag.

Return type bool

2.1.3 Submodules

2.1.4 pyplt.autoencoder module

class pyplt.autoencoder.**Autoencoder**(*input_size*, *code_size*, *encoder_topology*, *decoder_topology*, *activation_functions=None*, *learn_rate=0.001*, *error_threshold=0.001*, *epochs=10*, *batch_size=32*)
Bases: object

Autoencoder class.

Initialize the Autoencoder.

Parameters

- **input_size** (*int*) – the number of input features that will be fed into the network. This determines the number of neurons in the input layer (and therefore also in the output layer).
- **code_size** (*int*) – the number of neurons in the code layer (and therefore the size of the encoding).
- **encoder_topology** (*list of int*) – specifies the number of neurons in each layer of the encoder part of the network, excluding the input layer and the code layer.
- **decoder_topology** (*list of int*) – specifies the number of neurons in each layer of the decoder part of the network, excluding the code layer and the output layer.
- **activation_functions** (list of *pyplt.util.enums.ActivationType* or None, optional) – a list of the activation functions to be used across the neurons for each layer of the ANN (excluding input layer) (default None); if None, all layers will use the Rectified Linear Unit (ReLU) function i.e. *pyplt.util.enums.ActivationType.RELU*, except for the output layer which will use the Logisitic Sigmoid function i.e. *pyplt.util.enums.ActivationType.SIGMOID*.
- **learn_rate** (*float, optional*) – the learning rate used in the weight update step of the algorithm (default 0.1).
- **error_threshold** (*float, optional*) – a threshold at or below which the error of a model is considered to be sufficiently trained (default 0.1).
- **epochs** (*int, optional*) – the maximum number of iterations the algorithm should make over the entire set of training examples (default 10).
- **batch_size** (*int, optional*) – number of samples per gradient update (default 32).

clean_up()

Close the *tensorflow* session once the algorithm class instance is no longer needed.

V.IMP. THIS FUNCTION MUST BE CALLED WHEN THE CLASS INSTANCE IS NO LONGER IN USE unless a context manager is used around the Autoencoder class instance!!!

encode(*samples*, *progress_window=None*, *exec_stopper=None*)

Encode the given samples by running the given samples through the encoder part of the network.

Parameters

- **samples** (*array-like of shape n_samples x n_features*) – the samples to be input into the autoencoder.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the encoded sample – if execution is completed successfully.

- None – if experiment is aborted before completion by *exec_stopper*.

Return type array-like of shape n_samples x code_size

get_actfs()
Get the names of the activation functions of each layer in the network.

Returns the names of the activation functions of each layer.

Return type list of str

get_code_size()
Get the value of the code size parameter.

Returns the code size value.

Return type int

get_epochs()
Get the value of the epochs parameter.

Returns the epochs value.

Return type int

get_error_thresh()
Get the value of the error threshold parameter.

Returns the error threshold value.

Return type float

get_learn_rate()
Get the value of the learning rate parameter.

Returns the learning rate value.

Return type float

get_topology()
Get the topology of the network.

Returns the topology.

Return type list of int

get_topology_incl_input()
Get the topology of the network including the input layer.

Returns the topology including the input layer.

Return type list of int

init_train(*progress_window=None*, *exec_stopper=None*)
Initialize the model (topology).

This method is to be called if one wishes to initialize the model (topology) explicitly. This is done by declaring *tensorflow* placeholders, variables, and operations. If not called explicitly, the *train()* method will call it once implicitly.

Parameters

- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).

- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

predict(*samples*)
Run the given samples through the entire autoencoder, thus obtaining a reconstruction of the input.

Parameters **samples** (*array-like of shape n_samples x n_features*) – the samples to be input into the autoencoder.

Returns the autoencoder output for the given input samples.

Return type array-like of shape n_samples x n_features

train(*training_examples*, *progress_window=None*, *exec_stopper=None*)
Train the autoencoder.

Parameters

- **training_examples** (*array-like of shape n_samples x n_features*) – the input examples used to train the network.
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- training loss – if execution is completed successfully.
- None – if experiment is aborted before completion by *exec_stopper*.

2.1.5 pyplt.exceptions module

This module contains the definitions for several exceptions, errors and warnings occurring specifically in PLT.

exception pyplt.exceptions.**AutoencoderNormalizationValueError**(*norm_method*, *suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the normalization prior to use of autoencoder fails due to some non-numeric values.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters

- **norm_method** (*pyplt.util.enums.NormalizationType*) – the attempted normalization method.
- **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**DataSetValueWarning**
Bases: `UserWarning`

Custom warning to inform users that the data set they are loading contains values which are not entirely numerical (i.e., cannot be converted to float or int).

Extends `UserWarning`.

exception pyplt.exceptions.**FoldsRowsException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load a manual folds file with an invalid amount of rows.

Applies for KFoldCrossValidation.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class Exception (default False).

exception pyplt.exceptions.**FoldsSampleIDsException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load a manual folds file with the wrong sample IDs.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class Exception (default False).

exception pyplt.exceptions.**IDsException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load ranks containing anything other than IDs referring to objects.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class Exception (default False).

exception pyplt.exceptions.**IncompatibleFoldIndicesException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the amount of user-specified fold indices does not match the amount of samples in the dataset.

Applies for KFoldCrossValidation.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class Exception (default False).

exception pyplt.exceptions.**InvalidParameterValueException**(*parameter*, *value=None*, *method=None*, *is_algorithm=False*, *additional_msg=''*, *suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to use an invalid value for the given parameter of an algorithm/method.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters

- **parameter** (*str*) – the parameter for which the error was given.
- **value** (*object, optional*) – the value (assigned to *parameter*) causing the error (default None).
- **is_algorithm** (*bool, optional*) – specifies whether the parameter belonged to an algorithm (True) or method (False) (default False).
- **additional_msg** (*str, optional*) – an additional message to include in the exception message.
- **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**ManualFoldsFormatException**(*suppress=False*)

Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load a manual folds file with an invalid format.

A manual folds file is used to specify cross validation folds manually

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**MissingManualFoldsException**(*suppress=False*)

Bases: *pyplt.exceptions.PLTException*

Exception for when the user chooses to specify folds manually without uploading a manual folds file.

Applies for KFoldCrossValidation.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**NoFeaturesError**(*suppress=False*)

Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to run an experiment without any features to represent the objects.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**NoRanksDerivedError**(*suppress=False*)

Bases: *pyplt.exceptions.PLTException*

Exception for when no ranks could be derived from the given ratings-based dataset (single file format).

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**NonNumericFeatureException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load data containing non-numeric values.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**NonNumericValuesException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load data containing non-numeric values.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**NormalizationValueError**(*f_id*, *norm_method*, *f_name=None*, *suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts normalization on values that cannot be converted to int or float.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters

- **f_id** (*int*) – ID number of the feature causing the error.
- **norm_method** (*pyplt.util.enums.NormalizationType*) – the attempted normalization method.
- **f_name** (*str, optional*) – name of the feature causing the error (default None).
- **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**ObjectIDsFormatException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load objects containing non-numeric IDs.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**ObjectsFirstException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load ranks before first loading objects.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

exception pyplt.exceptions.**PLTException**(*summary*, *message*, *suppress=False*)
Bases: `Exception`

Base class for exceptions occurring in PLT.

Extends the class `Exception`.

Initializes the exception with a summary and message.

Parameters

- **summary** (*str*) – a very brief description of the exception.
- **message** (*str*) – an extended description of the exception.
- **suppress** (*bool, optional*) – specifies whether (False) or not (True) to call the parent constructor (default False).

get_message()
Get the message of the exception.

Returns the message of the exception.

Return type str

get_summary()
Get the summary of the exception.

Returns the summary of the exception.

Return type str

exception pyplt.exceptions.**ParamIgnoredWarning**
Bases: `UserWarning`

Custom warning to inform user that one of the parameters passed to a method or function is being ignored in favour of another parameter passed to the method/function which overrides the former.

Extends `UserWarning`.

exception pyplt.exceptions.**RanksFormatException**(*suppress=False*)
Bases: *pyplt.exceptions.PLTException*

Exception for when the user attempts to load ranks with an invalid format.

Extends *pyplt.exceptions.PLTException*.

Set the exception details.

Parameters **suppress** (*bool*) – specifies whether (False) or not (True) to call the constructor of parent class `Exception` (default False).

2.1.6 pyplt.experiment module

This module contains several backend classes and functions relating to the setting up a single experiment.

class pyplt.experiment.**Experiment**
Bases: `object`

Class encapsulating the set-up details of a single experiment.

get_autoencoder_details()
Get the details of the autoencoder used in the experiment (if applicable).

Returns a dict containing the autoencoder parameter names as its keys and the parameter values as its values.

Return type dict or None

get_autoencoder_loss()
Get the training loss of the autoencoder used in the experiment (if applicable).

Returns the training loss.

Return type float or None

get_data()
Get the loaded data.

If the single file format is used, a single *pandas.DataFrame* containing the data is returned. If the dual file format is used, a tuple containing both the objects and ranks (each a *pandas.DataFrame*) is returned.

Returns the loaded data.

Return type *pandas.DataFrame* or tuple of *pandas.DataFrame* (size 2)

get_features()
Get the features used in the experiment.

Returns the names of the features used in the experiment.

Return type list of str

get_norm_settings()
Get the normalization settings for each feature in the original objects data.

Returns a dict with the indices of the features as the dict's keys and the corresponding methods with which the features are to be normalized as the dict's values.

Return type dict of str (names of *pyplt.util.enums.NormalizationType*)

get_orig_features()
Get the original features used in the experiment.

If automatic feature extraction was enabled, this method will return the extracted features.

Returns the names of the features.

Return type list of str

get_pl_algorithm()
Get the preference learning algorithm used in the experiment.

Returns the preference learning algorithm used in the experiment.

Return type *pyplt.plalgorithms.base.PLAlgorithm*

get_time_meta_data()
Get meta data about the experiment related to time.

Returns a list containing the start timestamp (UTC), end timestamp (UTC), and duration of the experiment.

Return type list of float (size 3)

is_dual_format()
Indicate whether or not the data which has been loaded so far is in the dual file format.

Returns specifies whether the data is in the dual file format or not (single file format).

Return type bool

load_object_data(*file_path*, *has_fnames=False*, *has_ids=False*, *separator=','*, *col_names=None*, *na_filter=True*)

Attempt to load an objects file as specified by the user and carries out validation checks.

If the data fails a validation check, a `PLTException` is raised.

Parameters

- **file_path** (*str*) – the path of the file to be loaded.
- **has_fnames** (*bool, optional*) – specifies whether the file already contains feature names in the first row (default False).
- **has_ids** (*bool, optional*) – specifies whether the file already contains object IDs in the first column (default False).
- **separator** (*str, optional*) – the character separating items in the CSV file (default ‘,’).
- **col_names** (*list of str or None, optional*) – specifies the column names to be used (default None).
- **na_filter** (*bool, optional*) – specifies whether to detect missing value markers (default True).

Raises

- ***ObjectIDsFormatException*** – if one or more non-numeric object IDs are detected.
- ***NonNumericFeatureException*** – if the one or more feature with one or more non-numeric values are detected.

load_rank_data(*file_path*, *has_fnames=False*, *has_ids=False*, *separator=','*, *col_names=None*, *na_filter=True*)

Attempt to load a ranks file as specified by the user and carries out validation checks.

If the data fails a validation check, a `PLTException` is raised.

Parameters

- **file_path** (*str*) – the path of the file to be loaded.
- **has_fnames** (*bool, optional*) – specifies whether the file already contains feature names in the first row (default False).
- **has_ids** (*bool, optional*) – specifies whether the file already contains object IDs in the first column (default False).
- **separator** (*str, optional*) – the character separating items in the CSV file (default ‘,’).
- **col_names** (*list of str or None, optional*) – specifies the column names to be used (default None).
- **na_filter** (*bool, optional*) – a boolean indicating whether to detect missing value markers (default True).

Raises

- ***ObjectsFirstException*** – if the objects have not been loaded first.
- ***RanksFormatException*** – if the dataset contains an unexpected amount of columns.

- ***IDsException*** – if the dataset contains entries that do not refer to any object ID in the objects dataset.

load_single_data(*file_path*, *has_fnames=False*, *has_ids=False*, *separator=','*, *col_names=None*, *na_filter=True*, *mdm=0.0*, *memory='all'*)

Attempt to load a single file as specified by the user and carries out validation checks.

When the experiment is run, pairwise preferences are automatically derived based on the ratings (last column in the given dataset) of the given objects/samples. The dataset is thus split into objects and ranks. The derivation of the pairwise preferences/ranks may be controlled via the optional minimum distance margin (mdm) and memory arguments of this method. If the data fails a validation check, a `PLTException` is raised.

Parameters

- **file_path** (*str*) – the path of the file to be loaded.
- **has_fnames** (*bool, optional*) – specifies whether the file already contains feature names in the first row (default False).
- **has_ids** (*bool, optional*) – specifies whether the file already contains object IDs in the first column (default False).
- **separator** (*str, optional*) – the character separating items in the CSV file (default ‘,’).
- **col_names** (*list of str or None, optional*) – specifies the column names to be used (default None).
- **na_filter** (*bool, optional*) – a boolean indicating whether to detect missing value markers (default True).
- **mdm** (*float, optional*) – the minimum distance margin i.e., the minimum difference between the ratings of a given pair of objects/samples that is required for the object pair to be considered a valid and clear preference (default 0.0).
- **memory** (*int or 'all', optional*) – specifies how many neighbouring objects/samples are to be compared with a given object/sample when constructing the pairwise ranks (default ‘all’). If ‘all’, all objects/samples are compared to each other.

Raises

- ***ObjectIDsFormatException*** – if one or more non-numeric object IDs are detected.
- ***NonNumericFeatureException*** – if the one or more feature with one or more non-numeric values are detected.
- ***NoRanksDerivedError*** – if no pairwise preferences could be derived from the given data. This is either because there are no clear pairwise preferences in the data or because none of the clear pairwise preferences in the data conform to the chosen values for the rank derivation parameters (i.e., the minimum distance margin (*mdm*) and the memory (*memory*) parameters).
- ***InvalidParameterValueException*** – if the user attempts to use a negative value (i.e., smaller than 0.0) for the *mdm* parameter.

print_objects()

Print the objects data used in the experiment to console.

print_ranks()

Print the pairwise rank data used in the experiment to console.

run(*shuffle=False*, *random_state=None*, *debug=False*, *progress_window=None*, *exec_stopper=None*)
Run the the experiment: feature selection first (if applicable), then preference learning.

Prior to running feature selection and preference learning, this method applies all specified pre-processing steps (e.g., fold-splitting, rank derivation, normalization) to the loaded data (if applicable). The method also stores the experiment details and returns the results for further use.

Parameters

- **shuffle** (*bool, optional*) – specifies whether or not to shuffle the data (samples in the case of the single file format; ranks in the case of the dual file format) at the start of executing the experiment; i.e., prior to fold splitting, rank derivation, and normalization (if applicable) (default False).
- **random_state** (int or *numpy.random.RandomState*, optional) – seed for the random number generator (if int), or numpy RandomState object, used to shuffle the dataset if *shuffle* is True (default None).
- **debug** (*bool, optional*) – specifies whether or not to print notes to console for debugging purposes (default False).
- **progress_window** (*pyplt.gui.experiment.progresswindow.ProgressWindow*, optional) – a GUI object (extending the *tkinter.Toplevel* widget) used to display a progress log and progress bar during the experiment execution (default None).
- **exec_stopper** (*pyplt.util.AbortFlag*, optional) – an abort flag object used to abort the execution before completion (default None).

Returns

- the experiment results – if experiment is completed successfully.
  - eval_metrics – the resulting average train and, if applicable, average test accuracies
  - fold_metrics – the fold-specific start timestamp, end timestamp, evaluation metrics, and a *pandas.DataFrame* representation of the trained model.
- None – if aborted before completion by *exec_stopper*.

Return type

- eval_metrics – dict with keys:
  - '*Training Accuracy*'
  - '*Test Accuracy*' (if applicable)
- fold_metrics – list of tuple, each containing:
  - start_time – *datetime* timestamp (UTC timezone)
  - end_time – *datetime* timestamp (UTC timezone)
  - eval_metrics – dict with keys:
    - '*Training Accuracy*'
    - '*Test Accuracy*' (if applicable)
  - model – *pandas.DataFrame*

Raises

- ***NoFeaturesError*** – if there are no features/attributes in the objects data.

- ***NoRanksDerivedError*** – if rank derivation fails because no pairwise preferences could be derived from the given data. This is either because there are no clear pairwise preferences in the data or because none of the clear pairwise preferences in the data conform to the chosen values for the rank derivation parameters (i.e., the minimum distance margin (*mdm*) and the memory (*memory*) parameters).
- ***InvalidParameterValueException*** – if the user attempted to use a negative value (i.e., smaller than 0.0) for the *mdm* rank derivation parameter.
- ***NormalizationValueError*** – if normalization fails because one of the given values cannot be converted to int or float prior to the normalization.
- ***IncompatibleFoldIndicesException*** – if the amount of user-specified fold indices for cross validation does not match the amount of samples in the dataset.
- ***AutoencoderNormalizationValueError*** – if normalization prior to feature extraction via autoencoder fails due to the presence of non-numeric values in the dataset.

save_exp_log(*timestamp*, *path=''*)

Save a log of the experiment to a Comma Separated Value (CSV) file at the path indicated by the user.

The file contains several log items. The first column of the file contains the type of information presented by the given item while the second column contains the information itself.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **path** (*str, optional*) – the path at which the file is to be saved (default ""). If "", the file is saved to a logs folder in the project root directory by default.

save_model(*timestamp*, *fold_idx=0*, *path=''*)

Save the model or one of the models inferred in the experiment to file at the path indicated by the user.

The resulting file is of a Comma Separated Value (CSV) format.

Parameters

- **timestamp** (*float*) – the timestamp to be included in the file name.
- **fold_idx** (*int, optional*) – the index of the fold for which the model is to be saved (default 0). This parameter should only be used in the case of multiple folds (e.g., when K-Fold Cross Validation is used).
- **path** (*str, optional*) – the path at which the file is to be saved (default ""). If "", the file is saved to a logs folder in the project root directory by default.

set_autoencoder(*autoencoder*)

Set the autoencoder algorithm to be used to extract features from the dataset in the experiment.

Parameters **autoencoder** (*pyplt.autoencoder.Autoencoder* or None) – the given autoencoder algorithm.

set_fs_algorithm(*fs_algorithm*)

Set the preference learning algorithm used in the feature selection phase of the experiment to the given algorithm.

Parameters **fs_algorithm** (*pyplt.plalgorithms.base.PLAlgorithm*) – the given preference learning algorithm.

set_fs_evaluator(*fs_evaluator*)

Set the evaluation method used in the feature selection phase of the experiment to the given method.

Parameters **fs_evaluator** (*pyplt.evaluation.base.Evaluator*) – the given evaluation method.

set_fs_method(*fs_method*)
Set the feature selection method of the experiment to the given method.

Parameters **fs_method** (*pyplt.fsmethods.base.FeatureSelectionMethod*) – the given feature selection method.

set_normalization(*feature_ids*, *norm_method*)
Set the normalization method to be used for the given feature or features.

The actual application of the normalization method to the features occurs when the experiment is run.

N.B. If the dataset includes an object ID column as its first column, it is ignored by this method. Therefore, in such a case, an argument of 0 passed to the parameter *feature_ids* is taken to refer to the first feature in the dataset (the second column in the dataset) and not the object ID column.

Parameters

- **feature_ids** (*int or list of ints*) – the index of the feature or the list of features (columns in the dataset) for which the normalization method is to be set.
- **norm_method** (*pyplt.util.enums.NormalizationType*) – the normalization method to be used.

set_pl_algorithm(*pl_algorithm*)
Set the preference learning algorithm of the experiment to the given algorithm.

Parameters **pl_algorithm** (*pyplt.plalgorithms.base.PLAlgorithm*) – the given preference learning algorithm.

set_pl_evaluator(*pl_evaluator*)
Set the evaluation method of the experiment to the given method.

Parameters **pl_evaluator** (*pyplt.evaluation.base.Evaluator*) – the given evaluation method.

set_rank_derivation_params(*mdm=None*, *memory=None*)
Set the values of the parameters used during the derivation of ranks from ratings.

These only apply if a single file format is used.

Parameters

- **mdm** (*float or None, optional*) – the minimum distance margin i.e., the minimum difference between the ratings of a given pair of objects/samples that is required for the object pair to be considered a valid and clear preference (default None). If None, a value of 0.0 is used by default during rank derivation.
- **memory** (*int or 'all' or None, optional*) – specifies how many neighbouring objects/samples are to be compared with a given object/sample when constructing the pairwise ranks (default None). If None, a value of 'all' (i.e., all objects/samples are compared to each other) is used by default during rank derivation.

2.1.7 pyplt.experimentmanager module

This module contains the *ExperimentManager* class which enables the batching of experiments.

class pyplt.experimentmanager.**ExperimentManager**
Bases: object

Class for running a set of experiments in batch.

The user may add any number of experiments to the experiment list. The experiments may then be run in batch.

add_experiment(*experiment: pyplt.experiment.Experiment*)
Add an experiment to the list of experiments to be run in batch.

Parameters **experiment** (*pyplt.experiment.Experiment*) – the experiment to be added to the list.

run_all()
Run each of the experiments in the list sequentially.

Raises

- ***NoFeaturesError*** – if there are no features/attributes in the objects data of a given experiment.
- ***NoRanksDerivedError*** – if rank derivation fails because no pairwise preferences could be derived from the given data of a given experiment. This is either because there are no clear pairwise preferences in the data or because none of the clear pairwise preferences in the data conform to the chosen values for the rank derivation parameters (i.e., the minimum distance margin (*mdm*) and the memory (*memory*) parameters).
- ***InvalidParameterValueException*** – if the user attempted to use a negative value (i.e., smaller than 0.0) for the *mdm* rank derivation parameter of a given experiment.
- ***NormalizationValueError*** – if normalization fails for a given experiment because one of the given values cannot be converted to int or float prior to the normalization.

2.1.8 pyplt.main_gui module

This module runs the graphical user interface (GUI) of PLT.

The root widget of the GUI is a *pyplt.gui.mainmenu.MainMenu* widget. The rest of the GUI is managed by the *pyplt.gui* subpackage.

- Tutorial: How To Set Up And Run An Experiment
- Tutorial: How To Add An Algorithm Or Method

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

E

F

G

R

S

T

U

W

Z