# pdpbox Documentation

***Release 0.2.0+17.gb022a0a.dirty***

**SauceCat**

**Mar 14, 2021**

# Home

python partial dependence plot toolbox

CHAPTER 1

# Motivation

This repository is inspired by ICEbox. The goal is to visualize the impact of certain features towards model prediction for any supervised learning algorithm using partial dependence plots [R1] [R2]. PDPbox now supports all scikit-learn algorithms.

# CHAPTER 2

## The common headache

When using black box machine learning algorithms like random forest and boosting, it is hard to understand the relations between predictors and model outcome. For example, in terms of random forest, all we get is the feature importance. Although we can know which feature is significantly influencing the outcome based on the importance calculation, it really sucks that we don't know in which direction it is influencing. And in most of the real cases, the effect is non-monotonic. We need some powerful tools to help understanding the complex relations between predictors and model prediction.

CHAPTER 3

# Highlight

1. Helper functions for visualizing target distribution as well as prediction distribution.
2. Proper way to handle one-hot encoding features.
3. Solution for handling complex mutual dependency among features.
4. Support multi-class classifier.
5. Support two variable interaction partial dependence plot.

CHAPTER 4

# Documentation

- Latest version: http://pdpbox.readthedocs.io/en/latest/

CHAPTER 5

# Installation

- through pip:

```
$ pip install pdpbox
```

- through git:

```
$ git clone https://github.com/SauceCat/PDPbox.git
$ cd PDPbox
$ python setup.py install
```

## 5.1 References and Notes

### 5.1.1 References

### 5.1.2 Notes and Highlights

- One assumption made for the PDP is that the features in $X_C$ are uncorrelated with the features in $X_S$. If this assumption is violated, the averages, which are computed for the partial dependence plot, incorporate data points that are very unlikely or even impossible.

  For example, it's unreasonable to claim that height and weight is uncorrelated. If height is the feature to plot, only changing height through different values would create data points like someone is 2 meters but weighting below 50kg. Considering PDP is calculated by averaging through all data points, with these kind of unreasonable data points, the result might not be trustworthy. [R3]

  **Note:** check `data_transformer` parameter in `pdp_isolate` and `pdp_interact`.

- Some PD visualisations don't include the feature distribution. Omitting the distribution can be misleading, because you might over-interpret the line in regions, with almost no feature values. [R3]

  **Note:** check `plot_pts_dist` parameter in `pdp_plot`.

---

- There is one issue with ICE plots: It can be hard to see if the individual conditional expectation curves differ between individuals, because they start at different $\hat{f}(x)$. [R4]

  **Note:** check `center` parameters in `pdp_plot` and `pdp_interact_plot`.

---

- When many ICE curves are drawn the plot can become overcrowded and you don't see anything any more. [R4]

  **Note:** check `frac_to_plot` and `cluster` parameters in `pdp_plot` and `pdp_interact_plot`.

## 5.2 API Reference

### 5.2.1 pdpbox.info_plots.target_plot

pdpbox.info_plots.**target_plot**(*df*, *feature*, *feature_name*, *target*, *num_grid_points=10*, *grid_type='percentile'*, *percentile_range=None*, *grid_range=None*, *cust_grid_points=None*, *show_percentile=False*, *show_outliers=False*, *endpoint=True*, *figsize=None*, *ncols=2*, *plot_params=None*)

Plot average target value across different feature values (feature grids)

**Parameters**

**df: pandas DataFrame** data set to investigate on, should contain at least the feature to investigate as well as the target

**feature: string or list** feature or feature list to investigate, for one-hot encoding features, feature list is required

**feature_name: string** name of the feature, not necessary a column name

**target: string or list** column name or column name list for target value for multi-class problem, a list of one-hot encoding target column

**num_grid_points: integer, optional, default=10** number of grid points for numeric feature

**grid_type: string, optional, default='percentile'** 'percentile' or 'equal' type of grid points for numeric feature

**percentile_range: tuple or None, optional, default=None** percentile range to investigate for numeric feature when grid_type='percentile'

**grid_range: tuple or None, optional, default=None** value range to investigate for numeric feature when grid_type='equal'

**cust_grid_points: Series, 1d-array, list or None, optional, default=None** customized list of grid points for numeric feature

**show_percentile: bool, optional, default=False** whether to display the percentile buckets for numeric feature when grid_type='percentile'

**show_outliers: bool, optional, default=False** whether to display the out of range buckets for numeric feature when percentile_range or grid_range is not None

**endpoint: bool, optional, default=True** If True, stop is the last grid point Otherwise, it is not included

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**plot_params: dict or None, optional, default=None** parameters for the plot

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

**summary_df: pandas DataFrame** Graph data in data frame format

### Examples

Quick start with target_plot

```
from pdpbox import info_plots, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_target = test_titanic['target']
fig, axes, summary_df = info_plots.target_plot(
    df=titanic_data, feature='Sex', feature_name='Sex', target=titanic_target)
```

With One-hot encoding features

```
fig, axes, summary_df = info_plots.target_plot(
    df=titanic_data, feature=['Embarked_C', 'Embarked_Q', 'Embarked_S'],
    feature_name='Embarked', target=titanic_target)
```

With numeric features

```
fig, axes, summary_df = info_plots.target_plot(
    df=titanic_data, feature='Fare', feature_name='Fare',
    target=titanic_target, show_percentile=True)
```

With multi-class

```
from pdpbox import info_plots, get_dataset

test_otto = get_dataset.otto()
otto_data = test_otto['data']
otto_target = test_otto['target']
fig, axes, summary_df = info_plots.target_plot(
    df=otto_data, feature='feat_67', feature_name='feat_67',
    target=['target_0', 'target_2', 'target_5', 'target_8'])
```

### 5.2.2 pdpbox.info_plots.target_plot_interact

pdpbox.info_plots.**target_plot_interact**(*df*, *features*, *feature_names*, *target*, *num_grid_points=None*, *grid_types=None*, *percentile_ranges=None*, *grid_ranges=None*, *cust_grid_points=None*, *show_percentile=False*, *show_outliers=False*, *endpoint=True*, *figsize=None*, *ncols=2*, *annotate=False*, *plot_params=None*)

Plot average target value across different feature value combinations (feature grid combinations)

**Parameters**

**df: pandas DataFrame** data set to investigate on, should contain at least the feature to investigate as well as the target

**features: list** two features to investigate

**feature_names: list** feature names

**target: string or list** column name or column name list for target value for multi-class problem, a list of one-hot encoding target column

**num_grid_points: list, optional, default=None** number of grid points for each feature

**grid_types: list, optional, default=None** type of grid points for each feature

**percentile_ranges: list of tuple, optional, default=None** percentile range to investigate for each feature

**grid_ranges: list of tuple, optional, default=None** value range to investigate for each feature

**cust_grid_points: list of (Series, 1d-array, list), optional, default=None** customized list of grid points for each feature

**show_percentile: bool, optional, default=False** whether to display the percentile buckets for both feature

**show_outliers: bool, optional, default=False** whether to display the out of range buckets for both features

**endpoint: bool, optional** If True, stop is the last grid point, default=True Otherwise, it is not included

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**annotate: bool, default=False** whether to annotate the points

**plot_params: dict or None, optional, default=None** parameters for the plot

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

**summary_df: pandas DataFrame** Graph data in data frame format

#### Notes

- Parameters are consistent with the ones for function target_plot

- But for this function, you need to specify parameter value for both features in list format
- **For example:**
  - percentile_ranges = [(0, 90), (5, 95)] means
  - percentile_range = (0, 90) for feature 1
  - percentile_range = (5, 95) for feature 2

#### Examples

Quick start with target_plot_interact

```
from pdpbox import info_plots, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_target = test_titanic['target']

fig, axes, summary_df = info_plots.target_plot_interact(
    df=titanic_data, features=['Sex', ['Embarked_C', 'Embarked_Q', 'Embarked_S']],
    feature_names=['Sex', 'Embarked'], target=titanic_target)
```

### 5.2.3 pdpbox.info_plots.actual_plot

pdpbox.info_plots.**actual_plot**(*model*, *X*, *feature*, *feature_name*, *num_grid_points=10*, *grid_type='percentile'*, *percentile_range=None*, *grid_range=None*, *cust_grid_points=None*, *show_percentile=False*, *show_outliers=False*, *endpoint=True*, *which_classes=None*, *predict_kwds={}*, *ncols=2*, *figsize=None*, *plot_params=None*)

Plot prediction distribution across different feature values (feature grid)

**Parameters**

**model: a fitted sklearn model**

**X: pandas DataFrame** data set on which the model is trained

**feature: string or list** feature or feature list to investigate for one-hot encoding features, feature list is required

**feature_name: string** name of the feature, not necessary a column name

**num_grid_points: integer, optional, default=10** number of grid points for numeric feature

**grid_type: string, optional, default='percentile'** 'percentile' or 'equal', type of grid points for numeric feature

**percentile_range: tuple or None, optional, default=None** percentile range to investigate, for numeric feature when grid_type='percentile'

**grid_range: tuple or None, optional, default=None** value range to investigate, for numeric feature when grid_type='equal'

**cust_grid_points: Series, 1d-array, list or None, optional, default=None** customized list of grid points for numeric feature

**show_percentile: bool, optional, default=False** whether to display the percentile buckets, for numeric feature when grid_type='percentile'

**show_outliers: bool, optional, default=False** whether to display the out of range buckets for numeric feature when percentile_range or grid_range is not None

**endpoint: bool, optional** If True, stop is the last grid point, default=True Otherwise, it is not included

**which_classes: list, optional, default=None** which classes to plot, only use when it is a multi-class problem

**predict_kwds: dict, default={}** keywords to be passed to the model's predict function

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**plot_params: dict or None, optional, default=None** parameters for the plot

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

**summary_df: pandas DataFrame** Graph data in data frame format

### Examples

Quick start with actual_plot

```
from pdpbox import info_plots, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_features = test_titanic['features']
titanic_target = test_titanic['target']
titanic_model = test_titanic['xgb_model']
fig, axes, summary_df = info_plots.actual_plot(
    model=titanic_model, X=titanic_data[titanic_features],
    feature='Sex', feature_name='Sex')
```

With One-hot encoding features

```
fig, axes, summary_df = info_plots.actual_plot(
    model=titanic_model, X=titanic_data[titanic_features],
    feature=['Embarked_C', 'Embarked_Q', 'Embarked_S'], feature_name='Embarked')
```

With numeric features

```
fig, axes, summary_df = info_plots.actual_plot(
    model=titanic_model, X=titanic_data[titanic_features],
    feature='Fare', feature_name='Fare')
```

With multi-class

```
from pdpbox import info_plots, get_dataset

test_otto = get_dataset.otto()
otto_data = test_otto['data']
otto_model = test_otto['rf_model']
```

(continues on next page)

(continued from previous page)

```
otto_features = test_otto['features']
otto_target = test_otto['target']

fig, axes, summary_df = info_plots.actual_plot(
    model=otto_model, X=otto_data[otto_features],
    feature='feat_67', feature_name='feat_67', which_classes=[1, 2, 3])
```

### 5.2.4 pdpbox.info_plots.actual_plot_interact

pdpbox.info_plots.**actual_plot_interact**(*model*, *X*, *features*, *feature_names*, *num_grid_points=None*, *grid_types=None*, *percentile_ranges=None*, *grid_ranges=None*, *cust_grid_points=None*, *show_percentile=False*, *show_outliers=False*, *endpoint=True*, *which_classes=None*, *predict_kwds={}*, *ncols=2*, *figsize=None*, *annotate=False*, *plot_params=None*)

Plot prediction distribution across different feature value combinations (feature grid combinations)

**Parameters**

**model: a fitted sklearn model**

**X: pandas DataFrame** data set to investigate on, should contain at least the feature to investigate as well as the target

**features: list** two features to investigate

**feature_names: list** feature names

**num_grid_points: list, optional, default=None** number of grid points for each feature

**grid_types: list, optional, default=None** type of grid points for each feature

**percentile_ranges: list of tuple, optional, default=None** percentile range to investigate for each feature

**grid_ranges: list of tuple, optional, default=None** value range to investigate for each feature

**cust_grid_points: list of (Series, 1d-array, list), optional, default=None** customized list of grid points for each feature

**show_percentile: bool, optional, default=False** whether to display the percentile buckets for both feature

**show_outliers: bool, optional, default=False** whether to display the out of range buckets for both features

**endpoint: bool, optional** If True, stop is the last grid point, default=True Otherwise, it is not included

**which_classes: list, optional, default=None** which classes to plot, only use when it is a multi-class problem

**predict_kwds: dict, default={}** keywords to be passed to the model's predict function

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**annotate: bool, default=False** whether to annotate the points

**plot_params: dict or None, optional, default=None** parameters for the plot

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

**summary_df: pandas DataFrame** Graph data in data frame format

#### Notes

- Parameters are consistent with the ones for function actual_plot
- But for this function, you need to specify parameter value for both features in list format
- **For example:**
  - percentile_ranges = [(0, 90), (5, 95)] means
  - percentile_range = (0, 90) for feature 1
  - percentile_range = (5, 95) for feature 2

#### Examples

Quick start with actual_plot_interact

```
from pdpbox import info_plots, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_features = test_titanic['features']
titanic_target = test_titanic['target']
titanic_model = test_titanic['xgb_model']

fig, axes, summary_df = info_plots.actual_plot_interact(
    model=titanic_model, X=titanic_data[titanic_features],
    features=['Fare', ['Embarked_C', 'Embarked_Q', 'Embarked_S']],
    feature_names=['Fare', 'Embarked'])
```

### 5.2.5 pdpbox.pdp.PDPIsolate

**class** pdpbox.pdp.**PDPIsolate**(*n_classes*, *which_class*, *feature*, *feature_type*, *feature_grids*, *percentile_info*, *display_columns*, *ice_lines*, *pdp*, *count_data*, *hist_data*)

Save pdp_isolate results

**Parameters**

**n_classes: integer or None** number of classes for classifier, None when it is a regressor

**which_class: integer or None** for multi-class classifier, indicate which class the result belongs to

**feature: string or list** which feature is calculated on, list for one-hot encoding features

**feature_type: string** type of the feature

**feature_grids: list** feature grids

**percentile_info: list** percentile information for feature grids

**display_columns: list** columns to display as xticklabels

**ice_lines: pandas DataFrame** ICE lines

**pdp: 1-d numpy array** calculated PDP values

**count_data: pandas DataFrame** data points distribution

**hist_data: 1-d numpy array** data points distribution for numeric features

### 5.2.6 pdpbox.pdp.pdp_isolate

pdpbox.pdp.**pdp_isolate**(*model*, *dataset*, *model_features*, *feature*, *num_grid_points=10*, *grid_type='percentile'*, *percentile_range=None*, *grid_range=None*, *cust_grid_points=None*, *memory_limit=0.5*, *n_jobs=1*, *predict_kwds={}*, *data_transformer=None*)

Calculate PDP isolation plot

**Parameters**

**model: a fitted sklearn model**

**dataset: pandas DataFrame** data set on which the model is trained

**model_features: list or 1-d array** list of model features

**feature: string or list** feature or feature list to investigate, for one-hot encoding features, feature list is required

**num_grid_points: integer, optional, default=10** number of grid points for numeric feature

**grid_type: string, optional, default='percentile'** 'percentile' or 'equal', type of grid points for numeric feature

**percentile_range: tuple or None, optional, default=None** percentile range to investigate, for numeric feature when grid_type='percentile'

**grid_range: tuple or None, optional, default=None** value range to investigate, for numeric feature when grid_type='equal'

**cust_grid_points: Series, 1d-array, list or None, optional, default=None** customized list of grid points for numeric feature

**memory_limit: float, (0, 1)** fraction of memory to use

**n_jobs: integer, default=1** number of jobs to run in parallel. make sure n_jobs=1 when you are using XGBoost model. check: 1. https://pythonhosted.org/joblib/parallel.html#bad-interaction-of-multiprocessing-and-third-party-libraries 2. https://github.com/scikit-learn/scikit-learn/issues/6627

**predict_kwds: dict, optional, default={}** keywords to be passed to the model's predict function

**data_transformer: function or None, optional, default=None** function to transform the data set as some features changing values

**Returns**

**pdp_isolate_out: instance of PDPIsolate**

### 5.2.7 pdpbox.pdp.pdp_plot

pdpbox.pdp.**pdp_plot**(*pdp_isolate_out*, *feature_name*, *center=True*, *plot_pts_dist=False*, *plot_lines=False*, *frac_to_plot=1*, *cluster=False*, *n_cluster_centers=None*, *cluster_method='accurate'*, *x_quantile=False*, *show_percentile=False*, *figsize=None*, *ncols=2*, *plot_params=None*, *which_classes=None*)

Plot partial dependent plot

**Parameters**

**pdp_isolate_out: (list of) instance of PDPIsolate** for multi-class, it is a list

**feature_name: string** name of the feature, not necessary a column name

**center: bool, default=True** whether to center the plot

**plot_pts_dist: bool, default=False** whether to show data points distribution

**plot_lines: bool, default=False** whether to plot out the individual lines

**frac_to_plot: float or integer, default=1** how many lines to plot, can be a integer or a float

**cluster: bool, default=False** whether to cluster the individual lines and only plot out the cluster centers

**n_cluster_centers: integer, default=None** number of cluster centers

**cluster_method: string, default='accurate'** cluster method to use, default is KMeans, if 'approx' is passed, MiniBatchKMeans is used

**x_quantile: bool, default=False** whether to construct x axis ticks using quantiles

**show_percentile: bool, optional, default=False** whether to display the percentile buckets, for numeric feature when grid_type='percentile'

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**plot_params: dict or None, optional, default=None** parameters for the plot, possible parameters as well as default as below:

```
plot_params = {
    # plot title and subtitle
    'title': 'PDP for feature "%s"' % feature_name,
    'subtitle': "Number of unique grid points: %d" % n_grids,
    'title_fontsize': 15,
    'subtitle_fontsize': 12,
    'font_family': 'Arial',
    # matplotlib color map for ICE lines
    'line_cmap': 'Blues',
    'xticks_rotation': 0,
    # pdp line color, highlight color and line width
    'pdp_color': '#1A4E5D',
    'pdp_hl_color': '#FEDC00',
    'pdp_linewidth': 1.5,
    # horizon zero line color and with
    'zero_color': '#E75438',
    'zero_linewidth': 1,
    # pdp std fill color and alpha
    'fill_color': '#66C2D7',
    'fill_alpha': 0.2,
```

(continues on next page)

(continued from previous page)

```
    # marker size for pdp line
    'markersize': 3.5,
}
```

**which_classes: list, optional, default=None** which classes to plot, only use when it is a multi-class problem

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

### Examples

Quick start with pdp_plot

```
from pdpbox import pdp, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_target = test_titanic['target']
titanic_features = test_titanic['features']
titanic_model = test_titanic['xgb_model']

pdp_sex = pdp.pdp_isolate(model=titanic_model,
                          dataset=titanic_data,
                          model_features=titanic_features,
                          feature='Sex')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_sex, feature_name='sex')
```

With One-hot encoding features

```
pdp_embark = pdp.pdp_isolate(model=titanic_model, dataset=titanic_data,
                             model_features=titanic_features,
                             feature=['Embarked_C', 'Embarked_S', 'Embarked_Q'])
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_embark,
                         feature_name='Embark',
                         center=True,
                         plot_lines=True,
                         frac_to_plot=100,
                         plot_pts_dist=True)
```

With numeric features

```
pdp_fare = pdp.pdp_isolate(model=titanic_model,
                           dataset=titanic_data,
                           model_features=titanic_features,
                           feature='Fare')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_fare,
                         feature_name='Fare',
                         plot_pts_dist=True)
```

With multi-class

```
from pdpbox import pdp, get_dataset

test_otto = get_dataset.otto()
otto_data = test_otto['data']
otto_features = test_otto['features']
otto_model = test_otto['rf_model']
otto_target = test_otto['target']

pdp_feat_67_rf = pdp.pdp_isolate(model=otto_model,
                                 dataset=otto_data,
                                 model_features=otto_features,
                                 feature='feat_67')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_feat_67_rf,
                         feature_name='feat_67',
                         center=True,
                         x_quantile=True,
                         ncols=3,
                         plot_lines=True,
                         frac_to_plot=100)
```

### 5.2.8 pdpbox.pdp.PDPInteract

**class** pdpbox.pdp.**PDPInteract**(*n_classes*, *which_class*, *features*, *feature_types*, *feature_grids*, *pdp_isolate_outs*, *pdp*)

Save pdp_interact results

**Parameters**

**n_classes: integer or None** number of classes for classifier, None when it is a regressor

**which_class: integer or None** for multi-class classifier, indicate which class the result belongs to

**features: list** [feature1, feature2]

**feature_types: list** [feature1 type, feature2 type]

**feature_grids: list** [feature1 grid points, feature2 grid points]

**pdp_isolate_outs: list** [feature1 pdp_isolate result, feature2 pdp_isolate result]

**pdp: pandas DataFrame** calculated PDP values for each gird combination

### 5.2.9 pdpbox.pdp.pdp_interact

pdpbox.pdp.**pdp_interact**(*model*, *dataset*, *model_features*, *features*, *num_grid_points=None*, *grid_types=None*, *percentile_ranges=None*, *grid_ranges=None*, *cust_grid_points=None*, *memory_limit=0.5*, *n_jobs=1*, *predict_kwds={}*, *data_transformer=None*)

Calculate PDP interaction plot

**Parameters**

**model: a fitted sklearn model**

**dataset: pandas DataFrame** data set on which the model is trained

**model_features: list or 1-d array** list of model features

**features: list** [feature1, feature2]

**num_grid_points: list, default=None** [feature1 num_grid_points, feature2 num_grid_points]

**grid_types: list, default=None** [feature1 grid_type, feature2 grid_type]

**percentile_ranges: list, default=None** [feature1 percentile_range, feature2 percentile_range]

**grid_ranges: list, default=None** [feature1 grid_range, feature2 grid_range]

**cust_grid_points: list, default=None** [feature1 cust_grid_points, feature2 cust_grid_points]

**memory_limit: float, (0, 1)** fraction of memory to use

**n_jobs: integer, default=1** number of jobs to run in parallel. make sure n_jobs=1 when you are using XGBoost model. check: 1. https://pythonhosted.org/joblib/parallel.html#bad-interaction-of-multiprocessing-and-third-party-libraries 2. https://github.com/scikit-learn/scikit-learn/issues/6627

**predict_kwds: dict, optional, default={}** keywords to be passed to the model's predict function

**data_transformer: function or None, optional, default=None** function to transform the data set as some features changing values

**Returns**

**pdp_interact_out: instance of PDPInteract**

### 5.2.10 pdpbox.pdp.pdp_interact_plot

pdpbox.pdp.**pdp_interact_plot**(*pdp_interact_out*, *feature_names*, *plot_type='contour'*, *x_quantile=False*, *plot_pdp=False*, *which_classes=None*, *figsize=None*, *ncols=2*, *plot_params=None*)

PDP interact

**Parameters**

**pdp_interact_out: (list of) instance of PDPInteract** for multi-class, it is a list

**feature_names: list** [feature_name1, feature_name2]

**plot_type: str, optional, default='contour'** type of the interact plot, can be 'contour' or 'grid'

**x_quantile: bool, default=False** whether to construct x axis ticks using quantiles

**plot_pdp: bool, default=False** whether to plot pdp for each feature

**which_classes: list, optional, default=None** which classes to plot, only use when it is a multi-class problem

**figsize: tuple or None, optional, default=None** size of the figure, (width, height)

**ncols: integer, optional, default=2** number subplot columns, used when it is multi-class problem

**plot_params: dict or None, optional, default=None** parameters for the plot, possible parameters as well as default as below:

```
plot_params = {
    # plot title and subtitle
    'title': 'PDP interact for "%s" and "%s"',
    'subtitle': 'Number of unique grid points: (%s: %d, %s: %d)',
    'title_fontsize': 15,
    'subtitle_fontsize': 12,
```

(continues on next page)

(continued from previous page)

```
    # color for contour line
    'contour_color':  'white',
    'font_family': 'Arial',
    # matplotlib color map for interact plot
    'cmap': 'viridis',
    # fill alpha for interact plot
    'inter_fill_alpha': 0.8,
    # fontsize for interact plot text
    'inter_fontsize': 9,
}
```

**Returns**

**fig: matplotlib Figure**

**axes: a dictionary of matplotlib Axes** Returns the Axes objects for further tweaking

## Examples

Quick start with pdp_interact_plot

```
from pdpbox import pdp, get_dataset

test_titanic = get_dataset.titanic()
titanic_data = test_titanic['data']
titanic_target = test_titanic['target']
titanic_features = test_titanic['features']
titanic_model = test_titanic['xgb_model']

inter1 = pdp.pdp_interact(model=titanic_model,
                          dataset=titanic_data,
                          model_features=titanic_features,
                          features=['Age', 'Fare'],
                          num_grid_points=[10, 10],
                          percentile_ranges=[(5, 95), (5, 95)])
fig, axes = pdp.pdp_interact_plot(pdp_interact_out=inter1,
                                  feature_names=['age', 'fare'],
                                  plot_type='contour',
                                  x_quantile=True,
                                  plot_pdp=True)
```

With multi-class

```
from pdpbox import pdp, get_dataset

test_otto = get_dataset.otto()
otto_data = test_otto['data']
otto_features = test_otto['features']
otto_model = test_otto['rf_model']
otto_target = test_otto['target']

pdp_67_24_rf = pdp.pdp_interact(model=otto_model,
                                dataset=otto_data,
                                model_features=otto_features,
                                features=['feat_67', 'feat_24'],
                                num_grid_points=[10, 10],
```

(continues on next page)

(continued from previous page)

```
                                    percentile_ranges=[None, None],
                                    n_jobs=4)
fig, axes = pdp.pdp_interact_plot(pdp_interact_out=pdp_67_24_rf,
                                    feature_names=['feat_67', 'feat_24'],
                                    plot_type='grid',
                                    x_quantile=True,
                                    ncols=2,
                                    plot_pdp=False,
                                    which_classes=[1, 2, 3])
```

# Bibliography


[R1] Friedman, J. (2001). **Greedy Function Approximation: A Gradient Boosting Machine.** The Annals of Statistics, 29(5):1189–1232. (https://statweb.stanford.edu/~jhf/ftp/trebst.pdf)

[R2] Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E., **Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation.** (2015) Journal of Computational and Graphical Statistics, 24(1): 44-65 (https://arxiv.org/abs/1309.6392)

[R3] Christoph Molnar. (2018). **Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.** 5.1 Partial Dependence Plot (PDP) (https://christophm.github .io/interpretable-ml-book/pdp.html)

[R4] Christoph Molnar. (2018). **Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.** 5.2 Individual Conditional Expectation (ICE) (https://christophm.github.io/interpretable-ml-book/ice.html)

# Index

## A



## P



## T