pyCeterisParibus Documentation

Michał Kuźba

Apr 23, 2021

Contents:

sphinx-quickstart on Tue Jan 29 19:38:46 2019. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

CHAPTER 1

pyCeterisParibus

Please note that the Ceteris Paribus method is moved to the dalex Python package which is actively maintained. If you will experience any problem with pyCeterisParibus please consider the dalex implementation at https://dalex.drwhy.ai/python/api/.

pyCeterisParibus is a Python library based on an *R* package CeterisParibus. It implements Ceteris Paribus Plots. They allow understanding how the model response would change if a selected variable is changed. It's a perfect tool for What-If scenarios. Ceteris Paribus is a Latin phrase meaning all else unchanged. These plots present the change in model response as the values of one feature change with all others being fixed. Ceteris Paribus method is model-agnostic - it works for any Machine Learning model. The idea is an extension of PDP (Partial Dependency Plots) and ICE (Individual Conditional Expectations) plots. It allows explaining single observations for multiple variables at the same time. The plot engine is developed here.

1.1 Why is it so useful?

There might be several motivations behind utilizing this idea. Imagine a person gets a low credit score. The client wants to understand how to increase the score and the scoring institution (e.g. a bank) should be able to answer such questions. Moreover, this method is useful for researchers and developers to analyze, debug, explain and improve Machine Learning models, assisting the entire process of the model design.

1.2 Setup

Tested on Python 3.5+

PyCeterisParibus is on PyPI. Simply run:

```
pip install pyCeterisParibus
```

or install the newest version from GitHub by executing:

```
pip install git+https://github.com/ModelOriented/pyCeterisParibus
```

or download the sources, enter the main directory and perform:

```
https://github.com/ModelOriented/pyCeterisParibus.git
cd pyCeterisParibus
python setup.py install   # (alternatively use pip install .)
```

1.3 Docs

A detailed description of all methods and their parameters might be found in documentation.

To build the documentation locally:

```
pip install -r requirements-dev.txt
cd docs
make html
```

and open `_build/html/index.html`

1.4 Examples

Below we present use cases on two well-known datasets - Titanic and Iris. More examples e.g. for regression problems might be found here and in jupyter notebooks here.

Note, that in order to run the examples you need to install extra requirements from `requirements-dev.txt`.

1.5 Use case - Titanic survival

We demonstrate Ceteris Paribus Plots using the well-known Titanic dataset. In this problem, we examine the chance of survival for Titanic passengers. We start with preprocessing the data and creating an XGBoost model.

```
import pandas as pd
df = pd.read_csv('titanic_train.csv')

y = df['Survived']
x = df.drop(['Survived', 'PassengerId', 'Name', 'Cabin', 'Ticket'],
    inplace=False, axis=1)

valid = x['Age'].isnull() | x['Embarked'].isnull()
x = x[-valid]
y = y[-valid]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=42)
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

(continues on next page)

(continued from previous page)

```
# We create the preprocessing pipelines for both numeric and categorical data.
numeric_features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

categorical_features = ['Embarked', 'Sex']
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

```
from xgboost import XGBClassifier
xgb_clf = Pipeline(steps=[('preprocessor', preprocessor),
('classifier', XGBClassifier())])
xgb_clf.fit(X_train, y_train)
```

Here the pyCeterisParibus starts. Since this library works in a model agnostic fashion, first we need to create a wrapper around the model with uniform predict interface.

```
from ceteris_paribus.explainer import explain
explainer_xgb = explain(xgb_clf, data=x, y=y, label='XGBoost',
    predict_function=lambda X: xgb_clf.predict_proba(X)[:, 1])
```

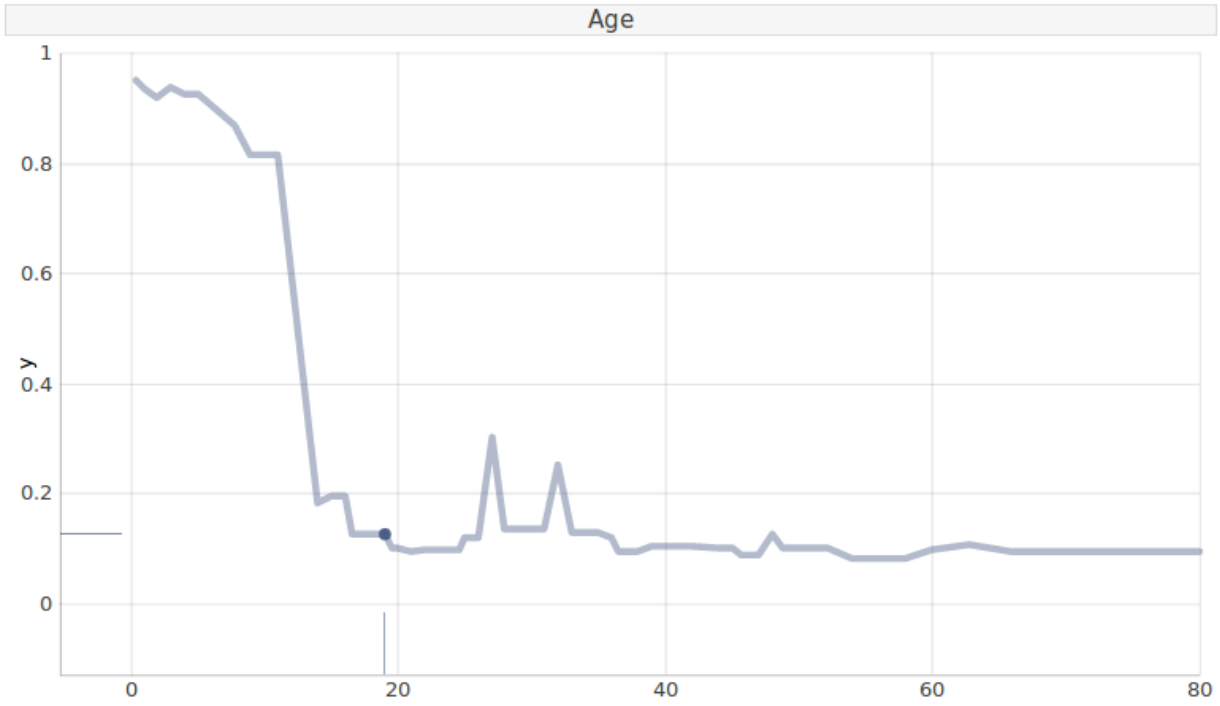
1.5.1 Single variable profile

Let's look at Mr Ernest James Crease, the 19-year-old man, travelling on the 3. class from Southampton with an 8 pounds ticket in his pocket. He died on Titanic. Most likely, this would not have been the case had Ernest been a few years younger. Figure 1 presents the chance of survival for a person like Ernest at different ages. We can see things were tough for people like him unless they were a child.

```
ernest = X_test.iloc[10]
label_ernest = y_test.iloc[10]
from ceteris_paribus.profiles import individual_variable_profile
cp_xgb = individual_variable_profile(explainer_xgb, ernest, label_ernest)
```

Having calculated the profile we can plot it. Note, that `plot_notebook` might be used instead of `plot` when used in Jupyter notebooks.

```
from ceteris_paribus.plots.plots import plot
plot(cp_xgb, selected_variables=["Age"])
```



1.5.2 Many models

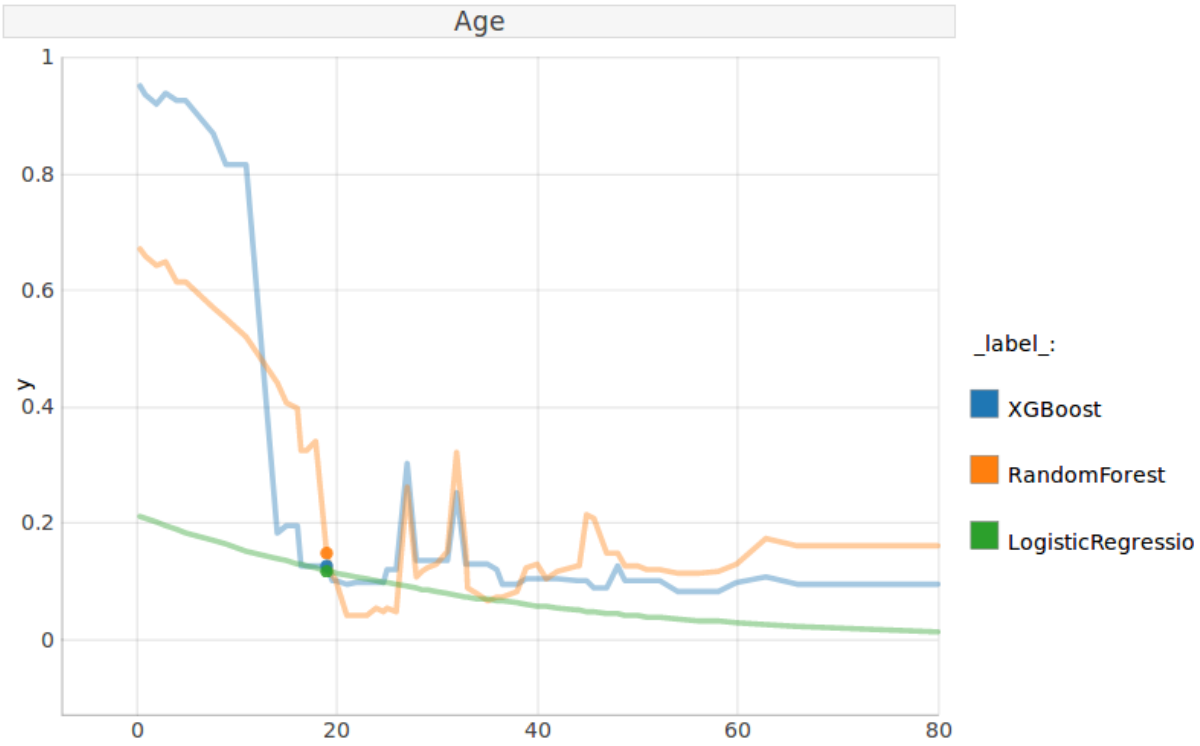
The above picture explains the prediction of XGBoost model. What if we compare various models?

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
rf_clf = Pipeline(steps=[('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())])
linear_clf = Pipeline(steps=[('preprocessor', preprocessor),
    ('classifier', LogisticRegression())])

rf_clf.fit(X_train, y_train)
linear_clf.fit(X_train, y_train)

explainer_rf = explain(rf_clf, data=x, y=y, label='RandomForest',
    predict_function=lambda X: rf_clf.predict_proba(X)[:, 1])
explainer_linear = explain(linear_clf, data=x, y=y, label='LogisticRegression',
    predict_function=lambda X: linear_clf.predict_proba(X)[:, 1])

plot(cp_xgb, cp_rf, cp_linear, selected_variables=["Age"])
```



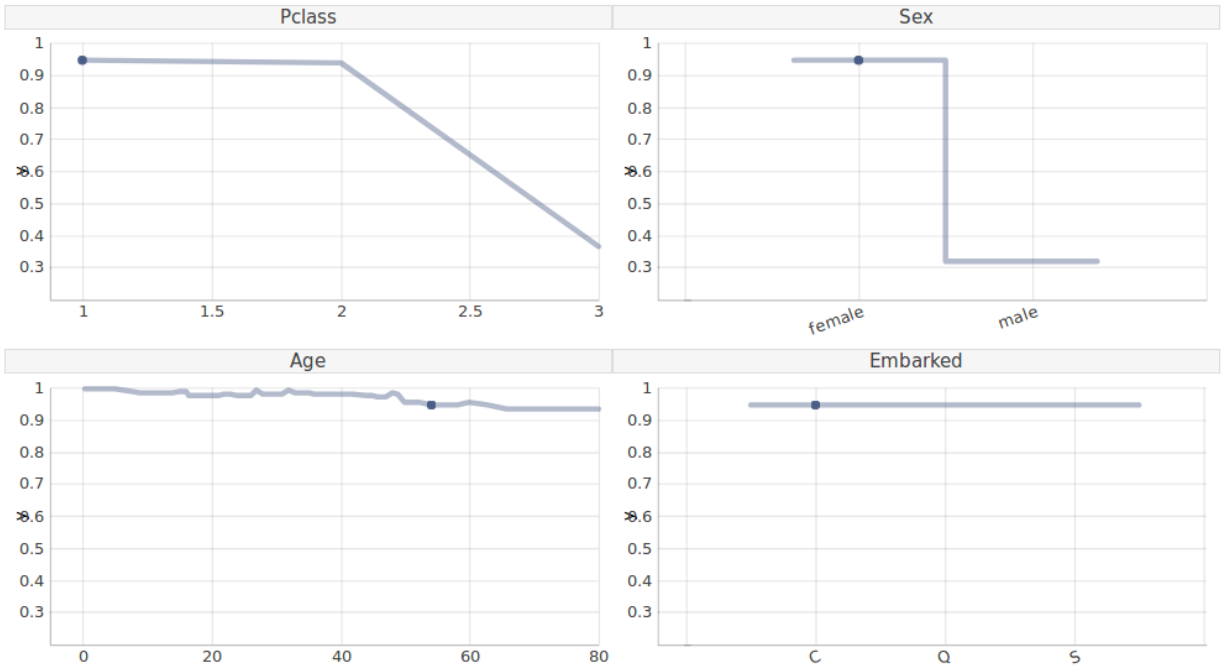
Clearly, XGBoost offers a better fit than Logistic Regression. Also, it predicts a higher chance of survival at child's age than the Random Forest model does.

1.5.3 Profiles for many variables

This time we have a look at Miss. Elizabeth Mussey Eustis. She is 54 years old, travels at 1. class with her sister Marta, as they return to the US from their tour of southern Europe. They both survived the disaster.

```
elizabeth = X_test.iloc[1]
label_elizabeth = y_test.iloc[1]
cp_xgb_2 = individual_variable_profile(explainer_xgb, elizabeth, label_elizabeth)
```

```
plot(cp_xgb_2, selected_variables=["Pclass", "Sex", "Age", "Embarked"])
```



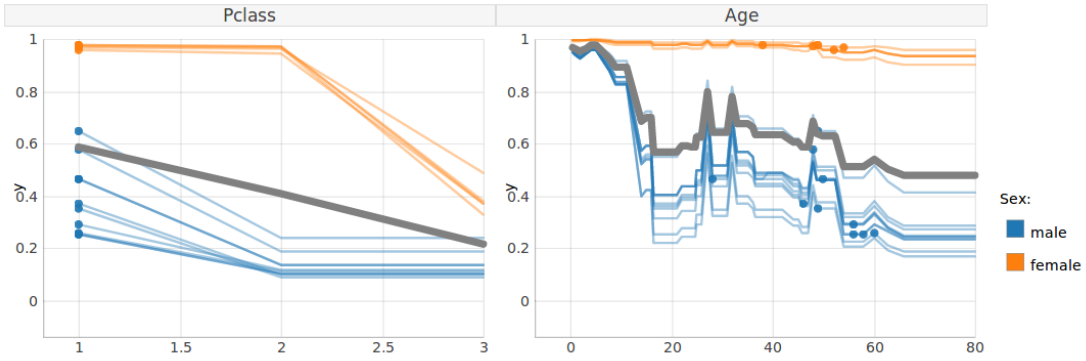
Would she have returned home if she had travelled at 3. class or if she had been a man? As we can observe this is less likely. On the other hand, for a first class, female passenger chances of survival were high regardless of age. Note, this was different in the case of Ernest. Place of embarkment (Cherbourg) has no influence, which is expected behaviour.

1.5.4 Feature interactions and average response

Now, what if we look at passengers most similar to Miss. Eustis (middle-aged, upper class)?

```
from ceteris_paribus.select_data import select_neighbours
neighbours = select_neighbours(X_train, elizabeth,
    selected_variables=['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'],
    n=15)
cp_xgb_ns = individual_variable_profile(explainer_xgb, neighbours)
```

```
plot(cp_xgb_ns, color="Sex", selected_variables=["Pclass", "Age"],
    aggregate_profiles='mean', size_pdps=6, alpha_pdps=1, size=2)
```



Dataset:

Search:

vname	Parch	_label_	_yhat_	_y_	Sex	_ids_	Fare	SibSp	Embarked	Pclass	Age
Age	0	XGBoost	0.4671548306941986		male	0	106.425	1	C	1	50
Age	0	XGBoost	0.6500056982040405		male	1	56.9292	1	C	1	49
Age	0	XGBoost	0.5780964493751526		male	2	76.7292	1	C	1	48
Age	0	XGBoost	0.9672768712043762		female	3	59.4	1	C	1	54
Age	0	XGBoost	0.9773577451705933		female	4	76.7292	1	C	1	49
Age	0	XGBoost	0.9554855227470398		female	5	78.2667	1	C	1	52
Age	0	XGBoost	0.9725086688995361		female	6	39.6	1	C	1	48
Age	0	XGBoost	0.9781637191772461		female	7	71.2833	1	C	1	38
Age	1	XGBoost	0.25963714718818665		male	8	79.2	1	C	1	60
Age	0	XGBoost	0.2908405065536499		male	9	35.5	0	C	1	56

Showing 1 to 105 of 105 entries

There are two distinct clusters of passengers determined with their gender, therefore a *PDP* average plot (on grey) does not show the whole picture. Children of both genders were likely to survive, but then we see a large gap. Also, being female increased the chance of survival mostly for second and first class passengers.

Plot function comes with extensive customization options. List of all parameters might be found in the documentation. Additionally, one can interact with the plot by hovering over a point of interest to see more details. Similarly, there is an interactive table with options for highlighting relevant elements as well as filtering and sorting rows.

1.5.5 Multiclass models - Iris dataset

Prepare dataset and model

```
iris = load_iris()

def random_forest_classifier():
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_model.fit(iris['data'], iris['target'])
    return rf_model, iris['data'], iris['target'], iris['feature_names']
```

Wrap model into explainers

```
rf_model, iris_x, iris_y, iris_var_names = random_forest_classifier()

explainer_rf1 = explain(rf_model, iris_var_names, iris_x, iris_y,
                        predict_function= lambda X: rf_model.predict_proba(X)[::, 0],␣
↪label=iris.target_names[0])
explainer_rf2 = explain(rf_model, iris_var_names, iris_x, iris_y,
                        predict_function= lambda X: rf_model.predict_proba(X)[::, 1],␣
↪label=iris.target_names[1])
explainer_rf3 = explain(rf_model, iris_var_names, iris_x, iris_y,
```

(continues on next page)

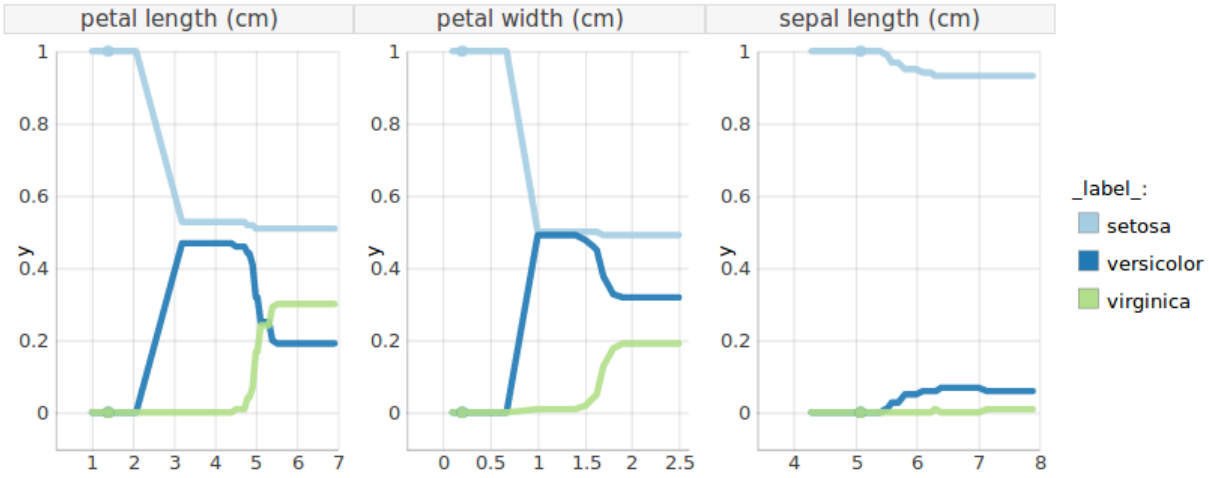
(continued from previous page)

```
                        predict_function= lambda X: rf_model.predict_proba(X)[:, 2],␣
↪label=iris.target_names[2])
```

Calculate profiles and plot

```
cp_rf1 = individual_variable_profile(explainer_rf1, iris_x[0], iris_y[0])
cp_rf2 = individual_variable_profile(explainer_rf2, iris_x[0], iris_y[0])
cp_rf3 = individual_variable_profile(explainer_rf3, iris_x[0], iris_y[0])

plot(cp_rf1, cp_rf2, cp_rf3, selected_variables=['petal length (cm)', 'petal width␣
↪(cm)', 'sepal length (cm)'])
```



1.6 Contributing

You're more than welcomed to contribute to this package. See the guideline.

1.7 Acknowledgments

Work on this package was financially supported by the 'NCN Opus grant 2016/21/B/ST6/0217'.

CHAPTER 2

ceteris_paribus

2.1 ceteris_paribus package

2.1.1 Subpackages

ceteris_paribus.plots package

Submodules

ceteris_paribus.plots.plots module

Module contents

2.1.2 Submodules

2.1.3 ceteris_paribus.explainer module

class ceteris_paribus.explainer.**Explainer**(*model*, *var_names*, *data*, *y*, *predict_fun*, *label*)

data
Alias for field number 2

label
Alias for field number 5

model
Alias for field number 0

predict_fun
Alias for field number 4

var_names
Alias for field number 1

y
Alias for field number 3

ceteris_paribus.explainer.**explain**(*model*, *variable_names=None*, *data=None*, *y=None*, *predict_function=None*, *label=None*)
This function creates a unified representation of a model, which can be further processed by various explainers

Parameters

- **model** – a model to be explained
- **variable_names** – names of variables, if not supplied then derived from data
- **data** – data that was used for fitting
- **y** – labels for the data
- **predict_function** – function that takes the data and returns predictions
- **label** – label of the model, if not supplied the function will try to infer it from the model object, otherwise unset

Returns Explainer object

2.1.4 ceteris_paribus.gower module

Gower Distance is a distance measure, that might be used to calculate the similarity between two observations with both categorical and numerical values. It also permits missing values in categorical variables. Therefore this measure might be applied in any dataset. Here we use it as a default function for finding the closest observations to the given one.

The original paper describing the idea might be found here.

This is the module for calculating gower's distance/dissimilarity

ceteris_paribus.gower.**gower_distances**(*data*, *observation*)
Return an array of distances between all observations and a chosen one Based on: https://sourceforge.net/projects/gower-distance-4python https://beta.vu.nl/nl/Images/stageverslag-hoven_tcm235-777817.pdf

2.1.5 ceteris_paribus.profiles module

class ceteris_paribus.profiles.**CeterisParibus**(*explainer*, *new_observation*, *y*, *selected_variables*, *grid_points*, *variable_splits*)
Bases: object

print_profile()

set_label(*label*)

split_by(*column*)
Split cp profile data frame by values of a given column

Returns sorted mapping of values to dataframes

ceteris_paribus.profiles.**individual_variable_profile**(*explainer*, *new_observation*, *y=None*, *variables=None*, *grid_points=101*, *variable_splits=None*)

Calculate ceteris paribus profile

Parameters

- **explainer** – a model to be explained
- **new_observation** – a new observation for which the profiles are calculated
- **y** – y true labels for *new_observation*. If specified then will be added to ceteris paribus plots
- **variables** – collection of variables selected for calculating profiles
- **grid_points** – number of points for profile
- **variable_splits** – dictionary of splits for variables, in most cases created with *_calculate_variable_splits()*. If None then it will be calculated based on validation data avaliable in the *explainer*.

Returns instance of CeterisParibus class

2.1.6 ceteris_paribus.select_data module

ceteris_paribus.select_data.**select_neighbours**(*data*, *observation*, *y=None*, *variable_names=None*, *selected_variables=None*, *dist_fun='gower'*, *n=20*)

Select observations from dataset, that are similar to a given observation

Parameters

- **data** – array or DataFrame with observations
- **observation** – reference observation for neighbours selection
- **y** – labels for observations
- **variable_names** – names of variables
- **selected_variables** – selected variables - require supplying variable names along with data
- **dist_fun** – 'gower' or distance function, as pairwise distances in sklearn, gower works with missing data
- **n** – size of the sample

Returns DataFrame with selected observations and pandas Series with corresponding labels if provided

ceteris_paribus.select_data.**select_sample**(*data*, *y=None*, *n=15*, *seed=42*)

Select sample from dataset.

Parameters

- **data** – array or dataframe with observations
- **y** – labels for observations
- **n** – size of the sample
- **seed** – seed for random number generator

Returns selected observations and corresponding labels if provided

2.1.7 Module contents

Python Module Index

C

Index