ExplainAI

by Feini Huang, Wei Shuangguan, Yongkun Zhang

Contact: huangfn3@mail2.sysu.edu.cn

Content

Overview

Installation

ExplainAI works in Python3.6+

Currently it requires scikit-learn0.22+

You can install ExplainAI using pip:

```
pip install ExplainAI
```

The latest version (ExplainAI 0.2.10)available for

```
pip install ExplainAI==0.2.10
```

Or clone codes from github:

https://github.com/HuangFeini/ExplainAI.git

In order to use the ExplainAI successfully, the following site-packages are required:

- pandas
- packaging
- psutil
- numpy
- sklearn
- scipy
- matplotlib
- numba

The latest ExplainAI 0.2.10 can work in

linux-Ubuntu 20.04+

Window 7+

Features

ExplainAI is a Python package which helps to visualize black-box machine learning and explain their predictions. It provides support for the following machine learning frameworks and functions:

- scikit-learn. Currently ExplainAI allows to explain predictions of scikit-learn regressors including DecisionTreeRegressor, LinearRegression, svm.SVR, KNeighborsRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, ExtraTreeRegressor， in order to show feature importances and feature effects.
- Post-hoc interpretation. Currently, ExplainAI integrated the following post-hoc methods: partial dependence plot (PDP), mean squared error (MSE)-based feature importance (MFI), permutation importance (PI), accumulated local effect (ALE), individual conditional expectation (ICE), local

interpretable model-agnostic explanations(LIME) and Shapley values. Details about each methods are given in the Feature effects section of the tutorial.
- Data preview. You can visualize observation and prediction distribution of feature or feature interaction, which are displayed in a figure of console.
- Two formats of explanation. You can upload your raw data (better in a csv) and after interpretation, you can get plot-based and text-based explanation in the console.
- Feature selection. The sequence backward selection (SBS) is provided. And some feature selection procedures specific for FLUXNET data also are available.

Basic usage

The basic usage involves following procedures:

1. upload your raw data and conduct data cleaning.
2. choose whether feature selection by sequential backward selection, if yes, a new input data is obtained, if no, you can use contrived work to select the features.
3. prediction, using sklearn model to train model and get prediction.
4. check the prediction and observation distribution.
5. interpretation. The trained model, input data matrix as input, the interpretation methods can be objectified.
6. display the results of interpretation (plot or text).

We recommend the users to accomplish step 1 to 3 due to their own requirements, and use the functions of step 4,5 provided in the ExpalinAI toolbox.

There are two main ways to interpret a black-box model:

1. inspect all the model predctions together and try to figure out how the model works globally;
2. inspect an individual prediction of a model, try to figure out why the model makes the decision it makes.

   For (1), ALE, PDP, MFI and PI, are all the avaliable "global" tools.

   For (2), ICE, Shapley values and LIME are all the avaliable "local" tools.

The interpretation are formatting in several ways, including figures, text, and a pandas Dataframe object. For example, a global interpretation are given as follows. You can also see these codes in the ExplainAI package/example/test.py

```
#1.upload your raw data and conduct data cleaning.
#here, we use the default dataset which has conducted data cleaning as an example.
from ExplainAI.flx_data.input import input_dataset
d=input_dataset(flag=0)
# d: the entire dataset (both input and output)




#2. choose whether feature selection by sequential backward selection, if yes, a new input
data is obtained, if no, you can use contrived work to select the features.
x=d.drop("SWC",axis=1)
f=list(x.columns)
y_ob=d['SWC']
```

```
y=y_ob

#3.prediction, using sklearn model to train model and get prediction.
#first, splite the data into training set and testing set
from sklearn.model_selection import train_test_split
xtr,xte,ytr,yte=train_test_split(x,y_ob,test_size=0.33)
#second, sklearn modeling
#here, you can use original sklearn function instead.
from ExplainAI.model.make_model import make_model
m,res,y_predict=make_model(modeltype='RandomForest',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
print(res)
#m: sklearn model object
#res: sklearn metrics
#y_predict: prediction values of testing set



#4.check the prediction and observation distribution.
from ExplainAI.preview import info_plots
import matplotlib.pyplot as plt
# show distribution with feature of interest ("TS")
fig1, axes, summary_df = info_plots.actual_plot(model=m, X=xte, feature="TS",
feature_name="TS")
fig2, axes, summary_df = info_plots.target_plot(df=d, target="SWC", feature="TS",
feature_name="TS")
# # show distribution under two features' interaction
fig3, axes, summary_df = info_plots.actual_plot_interact(model=m, X=xte, features=["DOY",
"TS"], feature_names=["DOY", "TS"])
fig4, axes, summary_df = info_plots.target_plot_interact(df=d, target="SWC", features=
["DOY", "TS"], feature_names=["DOY", "TS"])

# plot-based results
plt.show()
fig4.savefig('fig4.jpg')
#text-based results
print(summary_df)


#5.interpretation. The trained model, input data matrix as input, the interpretation
methods can be objectified.

#first, get input dataset and features
x=d.drop("SWC",axis=1)
f=list(x.columns)
y_ob=d['SWC']
y=y_ob
#second, interpretation (PI as an example)
from ExplainAI.explainers.pi.pi import permutation_importance_xai
rmse=res['RMSE']
```

```
p=permutation_importance_xai(m,f,x,y,rmse)

#6.display the results of interpretation (plot or text).
# plot-based results
#'pi.jpg' seemed in your save_path
#text-based results
print(p)
```

Why use ExplainAI?

At present, the post-hoc tools are widely used in many fields. However, it is not convenient to use different methods from different packages. Particularly, it leads to compatibility issues. To address this, ALE, PDP, ICE, Shapley, LIME, PI, MFI are integrated to one practical tool for ML developers and the decision-makers. Using ExplainAI, you can have a better experiences:

- you can call a ready-made function from ExplainAI and get a nicely formatted result immediately;
- formatting code can be reused between machine learning frameworks;
- algorithms like LIME try to explain a black-box model through a locally-fit simple, interpretable model. It means that with additional “simple” model supported algorithms like LIME will get more options automatically.

Tutorials

In this turoial, we will show how to use the ExplainAI using an example data set from a FLUXNET site or other two fixed format csv files. Users who want to build their own machine learning model can just jump to the feature effects section for the functions available to interpret and visualize the model.

Task

With the increasing demand for machine learning application in hydrometeorological forecast, we face the urge to demystify the black-box of machine learning as the lack of interpretability hampers adaptation of machine learning.

Here, taking soil moisture (SM) prediction of one FLUXNET site (Haibei,China, named as CH-Ha2) as an example, we used air forcing variables, timekeeping, energy processing, net ecosystem exchange and partitioning, and sundown as input data. We aimed to predict the daily SM via historial dataset. We aimed to interpret the model via ExplainAI toolbox.

Dataset

All dataset used in this tutorial is in the flx_data" dictionary of ExplainAI toolbox. The meta data of FLUXNET site data (Haibei,China, named as CH-Ha2) is available at https://ftp.fluxdata.org/.fluxnet_downloads_86523/FLUXNET2015/FLX_CN-Ha2_FLUXNET2015_FULLSET_2003-2005_1-4.zip

If users want to change the dataset, please modify the flag value of input_dataset(flag).

Filename	Content	Function
dataset.csv	Data after data processing and contrived work	data=input_dataset(flag=0)

```
from ExplainAI.flx_data.input import input_dataset
d=input_dataset(flag=0)
```

`:param` flag: index of which data set, see Tutorials section Dataset `:return` data: read data input, pandas.Dataframe

Black-box machine learning

For this version, sklean models are available.

```
# here is the given model list
from ExplainAI.model.make_model import make_model
model_list = ['DecisionTree', 'Linear', 'KNeighbors',
                  'RandomForest', 'AdaBoost',
                  'GradientBoosting', 'Bagging',
                  'BayesianRidge', 'SVR']

# obtain the dataset (training and testing)
x=d.drop("SWC",axis=1)
f=list(x.columns)
y_ob=d['SWC']
y=y_ob
from sklearn.model_selection import train_test_split
xtr,xte,ytr,yte=train_test_split(x,y_ob,test_size=0.33)

# model establishment
m,res,y_predict=make_model(modeltype='GradientBoosting',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
print(res)
#res:R2,MSE,MAE,RMSE
#m:trained model object
#y_predction:series, precdiction of testing set
```

If USER wants to modify the parameters of sklear models, please use the original sklearn model instead, for example:

```
from sklearn.metrics import
r2_score,mean_absolute_error,mean_squared_error,mean_squared_log_error
from sklearn import ensemble
m=ensemble.RandomForestRegressor(n_estimators=500)
m.fit(xtr,ytr)
y_predict = m.predict(xte)
res={"r2":r2_score(y_predict,yte),
             "MSE":mean_squared_error(y_predict,yte),
             "MAE":mean_absolute_error(y_predict,yte),
             "RMSE":mean_squared_log_error(y_predict,yte)}
print(res)
```

Preview

Data preview. You can visualize observation and prediction distribution of feature or feature interaction, which are displayed in a figure of console.

```
from ExplainAI.preview import *
info_plots.actual_plot(df, feature, feature_name, target, num_grid_points,
percentile_range, grid_range, cust_grid_points, show_percentile, show_outliers,endpoint,
figsize, ncols, plot_params)

info_plots.target_plot(df, feature, feature_name, target, num_grid_points,
percentile_range, grid_range, cust_grid_points, show_percentile, show_outliers,endpoint,
figsize, ncols, plot_params)

actual_plot_interact(df, feature, feature_name, target, num_grid_points, percentile_range,
grid_range, cust_grid_points, show_percentile, show_outliers,endpoint, figsize, ncols,
plot_params)

target_plot_interact(df, feature, feature_name, target, num_grid_points, percentile_range,
grid_range, cust_grid_points, show_percentile, show_outliers,endpoint, figsize, ncols,
plot_params)
```

`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

For example,

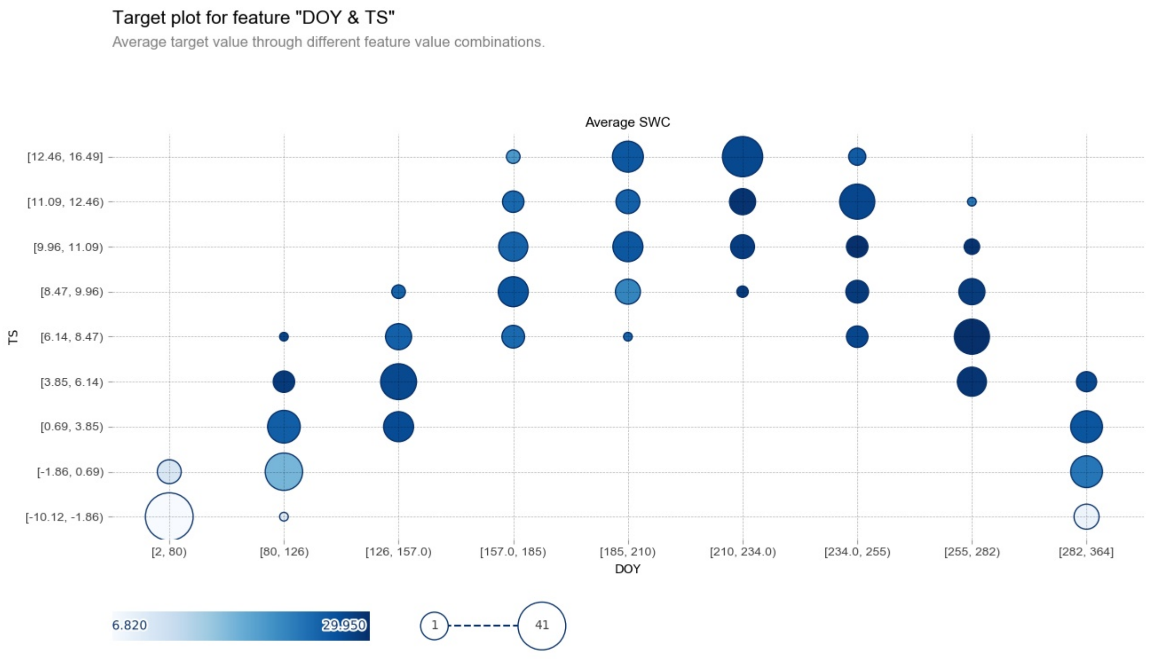
```python
# from preview import info_plots
# import matplotlib.pyplot as plt
from ExplainAI.preview import info_plots
import matplotlib.pyplot as plt
# show distribution with feature of interest ("TS")
fig1, axes, summary_df = info_plots.actual_plot(model=m, X=xte, feature="TS",
feature_name="TS")

fig2, axes, summary_df = info_plots.target_plot(df=d, target="SWC", feature="TS",
feature_name="TS")
# show distribution under two features' interaction
fig3, axes, summary_df = info_plots.actual_plot_interact(model=m, X=xte, features=["DOY",
"TS"], feature_names=["DOY", "TS"])

fig4, axes, summary_df = info_plots.target_plot_interact(df=d, target="SWC", features=
["DOY", "TS"], feature_names=["DOY", "TS"])


# plot-based results
plt.show()  # for windows
fig4.savefig('fig4.jpg') #for windows and linux
#text-based results
print(summary_df)
```

```
#print(summary_df)
x1  x2 display_column_1  ... value_upper_2  count       SWC
0    0   0          [1, 41)  ...     -7.009000   88.0  6.465909
1    0   1          [1, 41)  ...     -4.983333   32.0  6.775000
2    0   2          [1, 41)  ...     -1.191000    0.0  0.000000
3    0   3          [1, 41)  ...      0.436333    0.0  0.000000
4    0   4          [1, 41)  ...      3.735667    0.0  0.000000
..  ..  ..              ...  ...           ...    ...       ...
76   8   4       [325, 366]  ...      3.735667    0.0  0.000000
77   8   5       [325, 366]  ...      6.505000    0.0  0.000000
78   8   6       [325, 366]  ...      9.701000    0.0  0.000000
79   8   7       [325, 366]  ...     11.642333    0.0  0.000000
80   8   8       [325, 366]  ...     16.486000    0.0  0.000000
```



Feature effects

MSE-based Feature importance

Being one of the most pragmatic methods to quantify the feature importance, the Python package named as sklearn provides a specified importance evaluation for RF model. Note that R package named as randomForest also provides similar functions (Breiman, 2001). This method computes the importance from permuting out-of-bag data. First, for each tree, the MSE from prediction model on the out-of-bag portion of the training data is recorded. Next, this procedure is repeated for each feature.

Noted that, this method is specific-based, only for random forest.

```
mse_feature_importance()
```

`:param model:` sklearn model object, trained model

`:param data:` pd.Dataframe, input data

`:param target:`string, predicted target column name

`:param plot:`bool, if plt.show()

`:param top:`int, number of top feature at list

`:param save:`bool, if save the picture

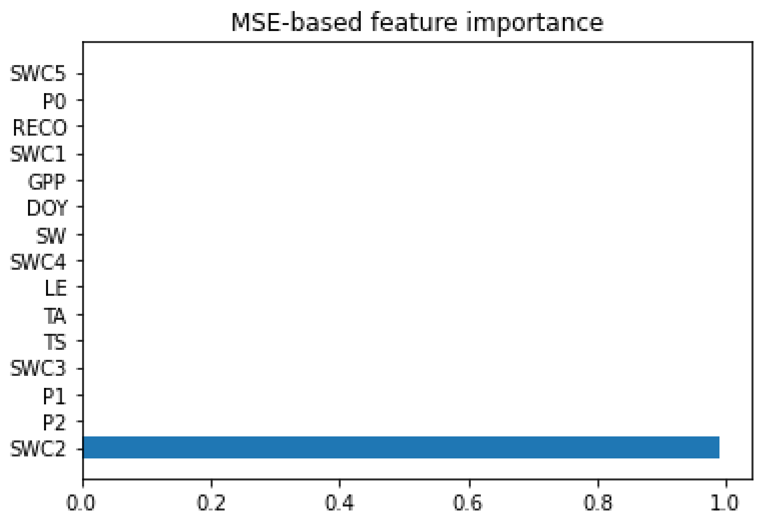
`:param save_path:`string, path of picture saved

`:return:` `df:` pd.Dataframe, MFI of features

For example,

```
from ExplainAI.explainers.mfi.mfi import mse_feature_importance
mfi=mse_feature_importance(model=m, data=d, target="SWC",top=15,save=True,plot=True)
print(mfi)
```

```
#print(mfi)
   feature       MFI
0      DOY  0.000376
1       TA  0.000674
2       SW  0.000427
3       TS  0.000689
4       LE  0.000547
5      GPP  0.000366
6     RECO  0.000237
7     SWC1  0.000258
8     SWC2  0.991041
9     SWC3  0.001017
10    SWC4  0.000514
11    SWC5  0.000157
12      P0  0.000193
13      P1  0.001379
14      P2  0.001950
```



Permutation importance

The PI of the observed importance provides a corrected measure of feature importance. PI computed with permutation importance are very helpful for deciding the significance of variables, and therefore improve model interpretability.

permutation_importance() offers an approach of PI storage in a dataframe and plot of ranking features.

```
permutation_importance()
```

`:param model:` sklearn model object, trained model

`:param features:` list or turple, fearture names storaged in a list or a turple

`:param plot:` bool, if plt.show()

`:param save:` bool, if save the picture

`:param save_path:` string, path of picture saved

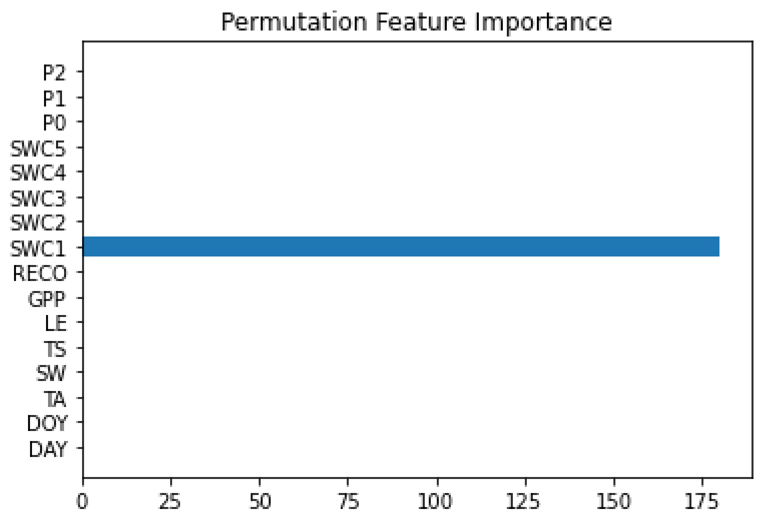
`:return:` pd.Dataframe, PI of features

For example,

```
from ExplainAI.explainers.pi.pi import permutation_importance_xai
rmse=res['RMSE']
permutation_importance_xai(m,f,x,y,rmse)
```

```
#print(p)
   feature          pi
0      DAY    0.419795
1      DOY    0.338754
2       TA    0.419098
3       SW    0.368461
4       TS    0.412025
5       LE    0.420868
```

```
6       GPP     0.434395
7      RECO     0.430843
8      SWC1   180.101879
9      SWC2     0.336018
10     SWC3     0.421370
11     SWC4     0.435304
12     SWC5     0.431914
13       P0     0.311173
14       P1     0.223754
15       P2     0.432301
```



Partial dependence plot

The PDP demonstrates the relationships between the features and predicted variable (Friedman, 2001). The PDP for regression is defined as:

$$p(x(s,j))(x(s,j)) = 1/n \sum^{n} f[x(s,j), x_c(i)]$$

where x(s,j) is the set of the feature of interest (as j-th feature) for which the partial dependence function should be plotted, p(x(s,j)) is the partial dependence value of j-th feature, n is the number of elements in x_s, and x_c is subset of other actual features values. PDP estimates the average marginal effect of predictors on the predicted SM, which can be a determined value in regression.

partial_dependence_plot_1d() provides one-dimentinal PDP.

```
explainers.partial_dependence_plot_1d()
```

`Parameters`

`model:` a fitted sklearn model

`data:` 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 `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

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

`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

`plot:` bool, if plt.show()

`save:` bool, if save the picture

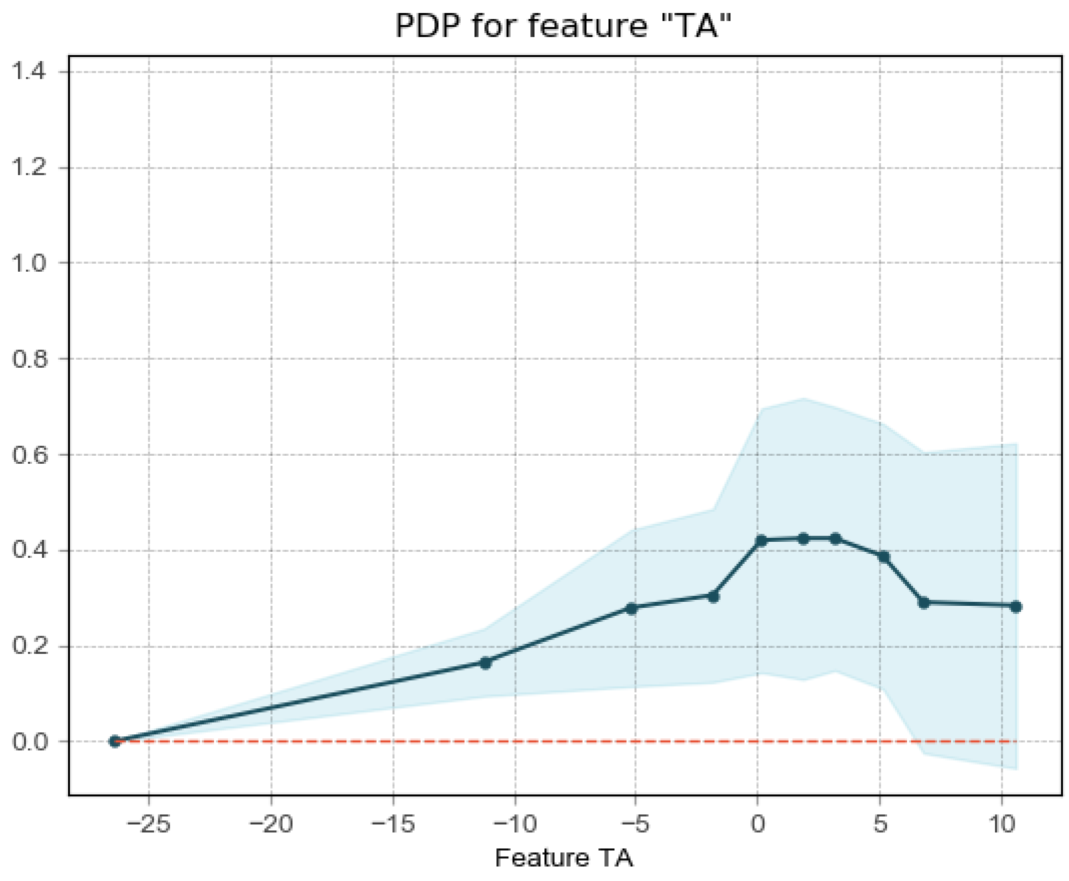
`save_path:` string, path of picture saved, default='pdp.jpg'

`return:` PDP dataframe

For example,

```
from ExplainAI.explainers.pdp.pdp import
partial_dependence_plot_1d,partial_dependence_plot_2d
pd1=partial_dependence_plot_1d(model=m,data=x,model_features=f,feature="TA",plot=True,save=
True)
print(pd1)
```

```
   x       xticklabels  count  count_norm
0  0  [-26.43, -11.24)     53    0.110879
1  1   [-11.24, -5.18)     53    0.110879
2  2    [-5.18, -1.82)     53    0.110879
3  3     [-1.82, 0.17)     53    0.110879
4  4      [0.17, 1.88)     53    0.110879
5  5      [1.88, 3.18)     53    0.110879
6  6      [3.18, 5.13)     53    0.110879
7  7       [5.13, 6.8)     53    0.110879
8  8       [6.8, 10.6]     54    0.112971
```



partial_dependence_plot_2d() provides two-dimentinal PDP, of which the two features have no interactions.

```
explainers.partial_dependence_plot_2d()
```

`parameter:`

`model:` a fitted sklearn model

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

`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

`plot:` bool, if plt.show()

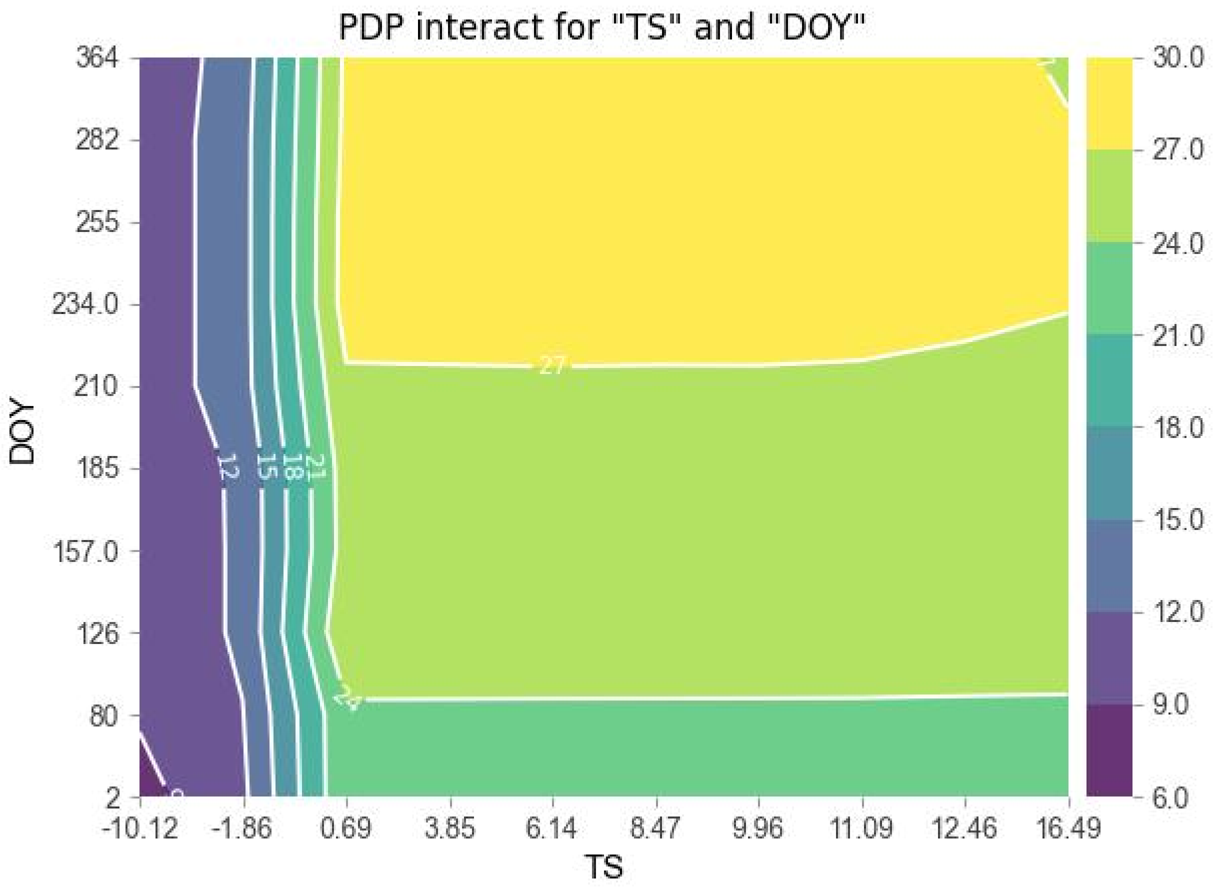
`save:` bool, if save the picture

`save_path:` string, path of picture saved, default='pdp.jpg'

`return:` PDP dataframe

```python
from ExplainAI.explainers.pdp.pdp import partial_dependence_plot_2d
pd2=partial_dependence_plot_2d(model=m,data=x,model_features=f,features=
["TS",'DOY'],plot=True,save=True)
print(pd2)
```

```
#print(pd2)
        TS    DOY      preds
0  -10.120    2.0   8.234782
1  -10.120   80.0   8.984797
2  -10.120  126.0  10.730751
3  -10.120  157.0  10.660559
4  -10.120  185.0  10.667151
..     ...    ...        ...
95  16.486  210.0  26.322354
96  16.486  234.0  26.973159
97  16.486  255.0  26.940115
98  16.486  282.0  26.804992
99  16.486  364.0  25.354090
```



Individual conditional expectation

ICE was proposed by Goldstein et al. (2015). The ICE concept is given by:

$$p(x(s,j)) = f(x(s,j), xc)$$

For a feature of interest, ICE plots highlight the variation in the fitted values across the range of covariate. In other words, the ICE provides the plots of dependence of the predicted response on a feature for each instance separately.

```
individual_conditional_exception()
```

`:param data:` pandas.DataFrame, the sample data from which to generate ICE curves

`:param column:` str, the name of the column in `data` that will be varied to generate ICE curves

`:param predict:` callable, the function that generates predictions from the model.

`:param num_grid_points:` None or int,the number of grid points to use for the independent

`:param frac_to_plot:` float, the fraction of ICE curves to plot.

`:param plot_points:` bool, whether or not to plot the original data points on the ICE curves.

`:param x_quantile:` bool, if `True`, the plotted x-coordinates are the quantiles of `ice_data.index`

`:param plot_pdp:` if `True`, plot the partial depdendence plot. In this case, `pdp_kwargs` is passed as keyword arguments to `plot`.

`:param centered:` if `True`, each ICE curve is centered to zero at the percentile closest to `centered_quantile`.

`:param color_by:` If a string, color the ICE curve by that level of the column index.

`:param cmap:` matplotlib Colormap

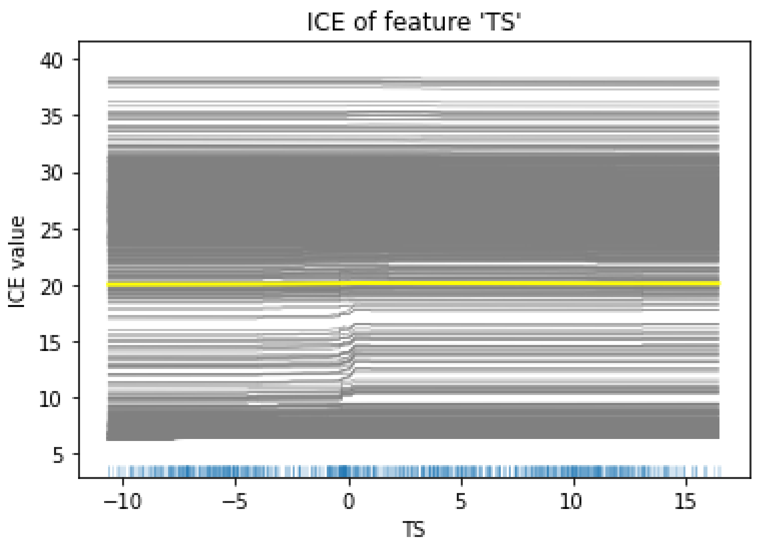
`:param ax:`: `None` or `matplotlib` `Axes`,the `Axes` on which to plot the ICE curves

`plot:` bool, if plt.show() `save:` bool, if save the picture `save_path:` string, path of picture saved, default='pdp.jpg'

`retuen:` Dataframe, ICE data

For example,

```
from ExplainAI.explainers.ice.ice import individual_conditional_exception
i=individual_conditional_exception(data=x, feature='TS',
model=m,plot=True,save=True,save_path='ice.jpg')
i.to_csv('ice.csv')
```



Accumulated Local Effect

The ALE is a more sophisticated method to evaluate the feature effects, owing to averaging the differences in the prediction model for conditional distribution (Apley et al., 2019). One-dimensional ALE (1D ALE) shows the dominate effects with the feature of interest variation.

$$f_j(x) = \sum^{h} 1/(n_j(k)) \sum\nolimits_m [f(z(k,j), x_(i, \backslash\mathrm{j})]) - f(z(k-1,j), x(i, \backslash\mathrm{j})) - c$$

$$h = k_j(k)$$

$$m = (i : x(i,j) \in N_j(k))$$

And the constant c is calculated to make sure the following equation:

$$1/n \sum^{n} f_j(x) = 0$$

where fj donates the ALE values and it visualizes the main effect dependence of modelling on x_j, x(i,\j)=(x(i,l):l=1,...,d;l≠j), where the subscript \j means all feature but the j-th. Similarly, Nj (k)=(z(k-1,j),z(k,j);k=1,2,...,K) donates a sufficiently fine partition of the sample range of x(i,j) into K intervals.

accumulated_local_effect_1d() offers one-dimentional ALE.

```
accumulated_local_effect_1d()
```

`parameter:`

`model :`object or function A Python object that contains 'predict' method. It is also possible to define a custom prediction function with 'predictor' parameters that will override 'predict' method of model.

`train_set :`pandas DataFrame Training set on which model was trained.

`features :` string or tuple of string A single or tuple of features' names.

`bins :` int Number of bins used to split feature's space.

`monte_carlo :` boolean Compute and plot Monte-Carlo samples.

`predictor :` function Custom function that overrides 'predict' method of model.

`monte_carlo_rep :`int Number of Monte-Carlo replicas.

`monte_carlo_ratio :` float Proportion of randomly selected samples from dataset at each Monte-Carlo replica.

`plot:`bool, if plt.show()

`save:`bool, if save the picture

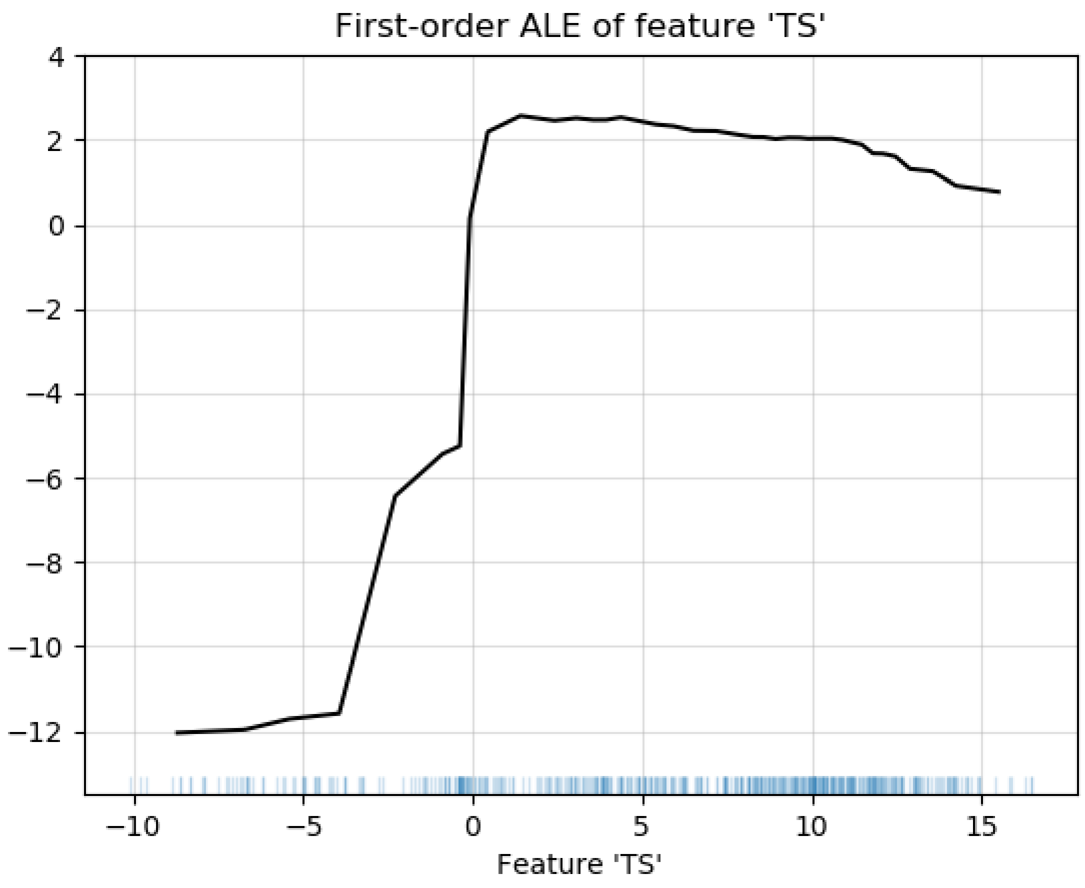
`save_path:`string, path of picture saved, default='pdp.jpg'

`retuen:`Dataframe, ALE data

For example,

```
from ExplainAI.explainers.ale.ale import accumulated_local_effect_1d
a1=accumulated_local_effect_1d(model=m, train_set=x,
features='TA',plot=False,save=True,monte_carlo=False)
print(a1)
```

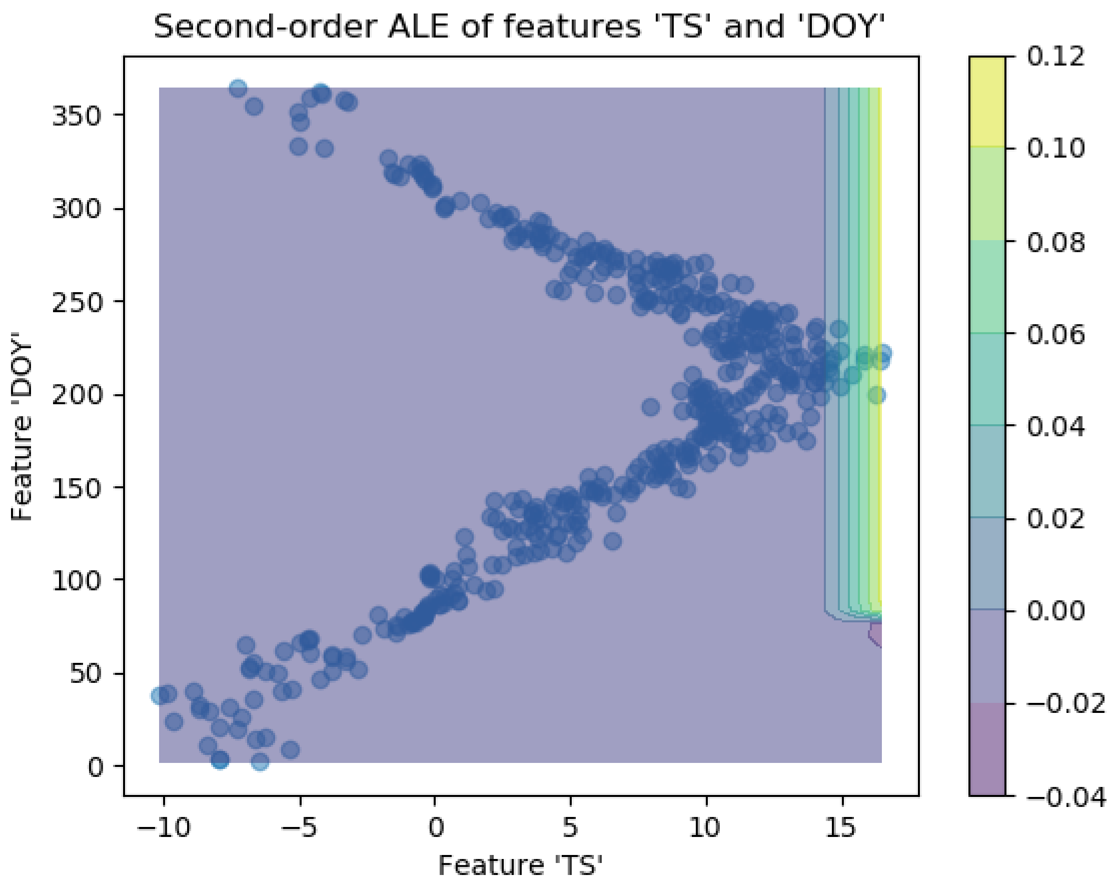
```
#print(a1)
    quantiles        ALE
0   -7.266475 -12.246686
1   -6.201750 -12.157186
2   -4.612375 -11.906602
3   -3.260400 -11.857769
4   -1.304625  -5.738269
5   -0.470650  -5.120852
..    ...
36  13.149300   1.387807
37  13.976050   1.356391
38  14.514150   1.266307
39  16.486000   1.198035
```



Two-dimensional ALE (2D ALE) solely displays the additional effect of an interaction between two features, which does not contain the main effect of each feature.

```
from ExplainAI.explainers.ale.ale import accumulated_local_effect_2d
a2=accumulated_local_effect_2d(m, train_set=x, features=['TS', 'DOY'], plot=False,
bins=40,save=True)
```

```
         -10.6360  -9.0075   -8.1510   ...   13.1090   14.1425   16.4860
1.000    0.000119  0.000119  0.000119  ...  0.000119  0.000119  0.000119
10.000   0.000119  0.000119  0.000119  ...  0.000119  0.000119  0.000119
...
338.000  0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
347.250  0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
356.625  0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
366.000  0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
```



Shapley values

Considering the all-possible interactions and redundancies between features, all combinations of features are tested. Apart from the evaluation for the training set, the Shapley values method can be applied on any data subset or even a single instance (Shapley and Roth, 1988). The Shapley values of a feature value is its contribution to the predicted result, weighted and summed over all possible feature value combinations (Štrumbelj and Kononenko, 2013):

$$\varphi(i,j)(val) = \sum_S |S|!(p-|S|-1)!/p![val(S \cup x(i,j)) - val(S)]$$

$$S \subseteq [x(i,1), \ldots, x(i,p)] \setminus x(i,j)$$

where S is a subset of the features used in an alliance, x_i is the vector of feature value of interest of instance j, p donates the number of features, and val is the prediction for feature values in subset S that are marginalized over features that are not included in subset S.

Here are the two versions of Shapley values in different operating systems. For windows only, the pictures would be captured in the console which requires the manual saving. For Linux and Windows, you can choose your save path to save the pictures.

At first, the Shapley values function is treated as a vessel.

```
shap_obj=shap_func()
```

`:param model:` sklearn model object

`:param x:` dataframe, input feature dataset

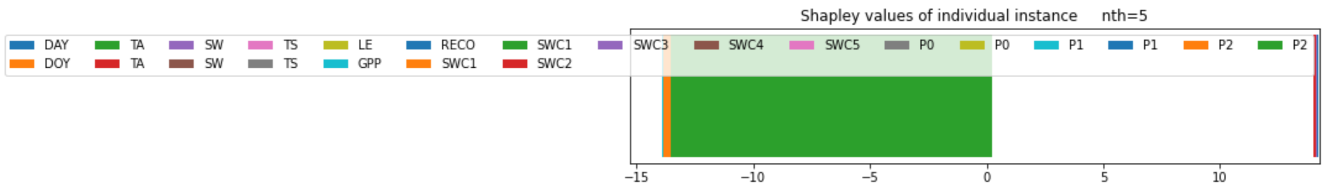
`:param features:` list, feature names

record_shap() can export calculated shapley values in "shap.csv".

```
from ExplainAI.explainers.shap_func.shap_func import shap_func
ss=shap_func(m,x)
ss.record_shap()
```

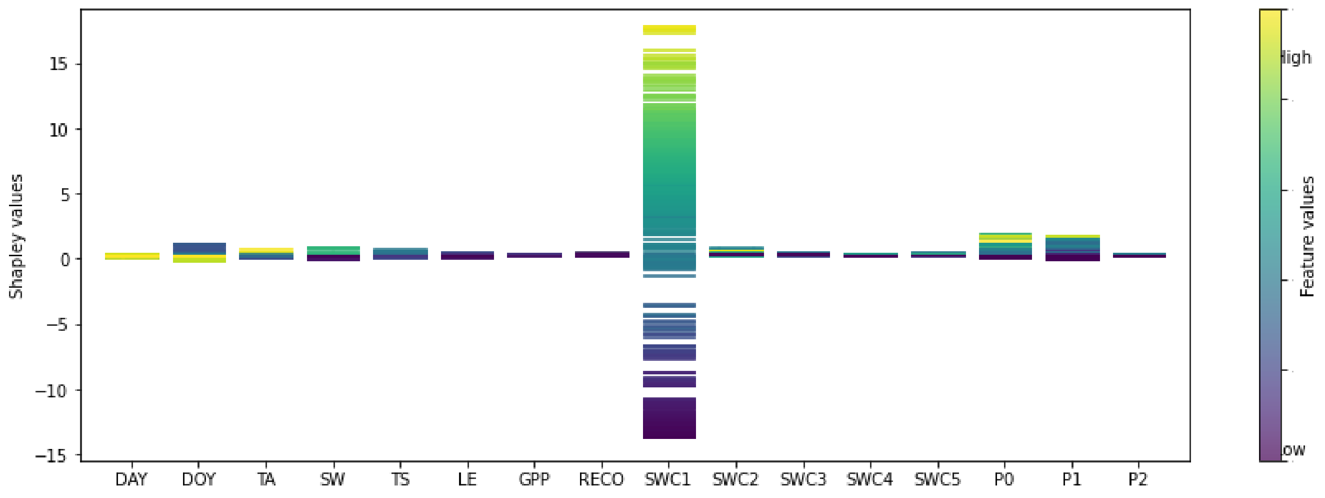
single_shap() offers shapley values for an individual instance.

```
ss.single_shap(nth=6)
#nth donates sequence of instance of interest.
```



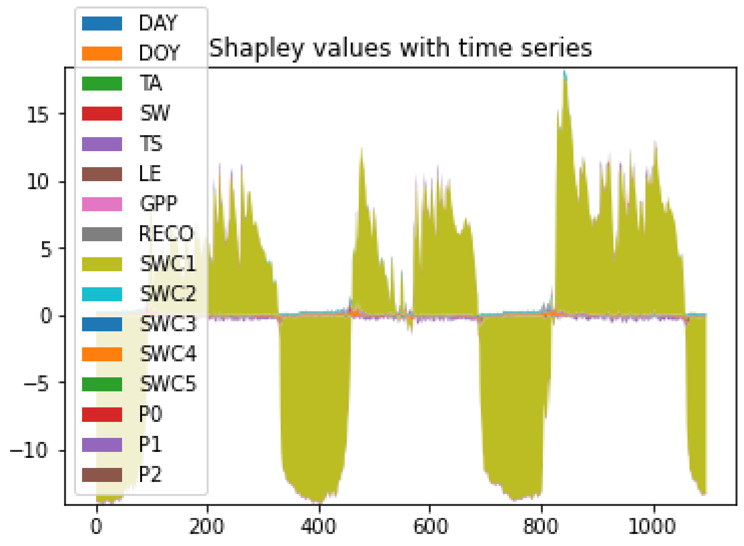
feature_value_shap() provides shapley values distribution with feature values.

```
ss.feature_value_shap()
```



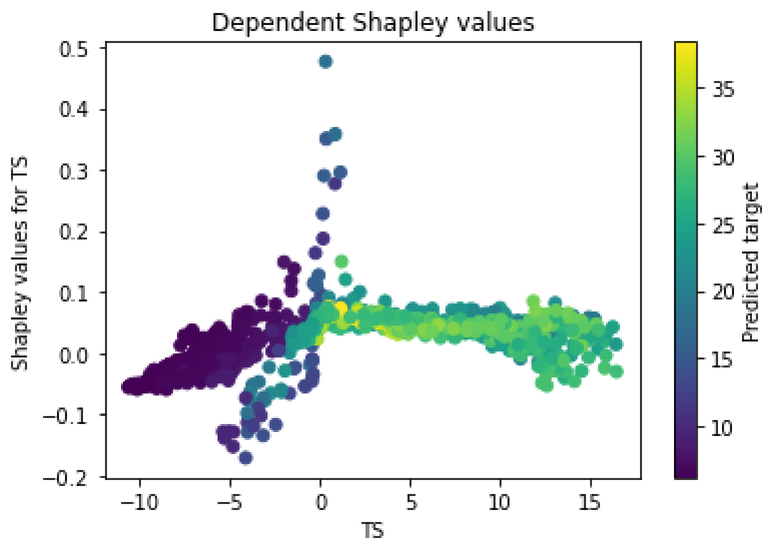
time_shap() provides Shapley values distribution with time series.

```
ss.time_shap()
```



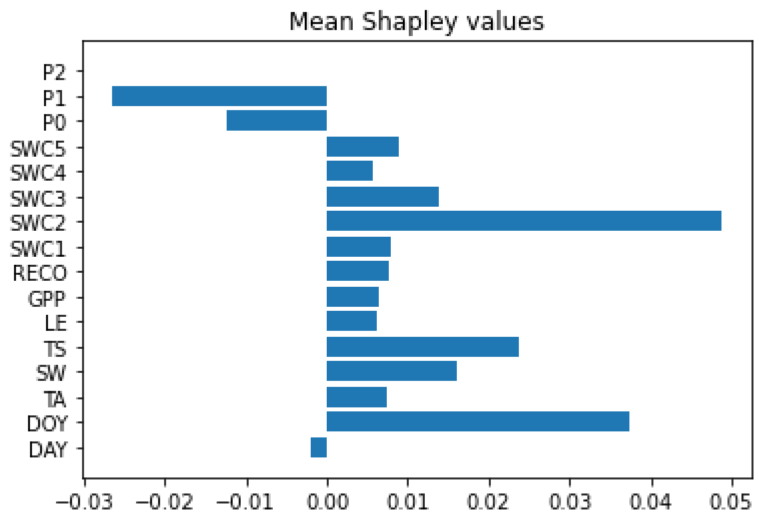
depend_shap() provides Shapley values distribution with feature variation.

```
ss.depend_shap(depend_feature='TS')
```



mean_shap() provides averaged Shapley values of features.

```
ss.mean_shap()
```



Local Interpretable Model-Agnostic Explanations

Local Interpretable Model-Agnostic Explanations (LIME) is an attempt to make these complex models at least partly understandable (Ribeiro, et al., 2016). Generally, the surrogate model after training, aims to approximate the predictions of the underlying black box model.

```
lime_explainations()
```

`:param model:` sklearn model object

`:param train_data:` dataframe, input feature dataset

`:param features:` list, feature names

`:param target:` string, target feature name

`:param instance_sequence:` int, instance number

`:param num_features:` int, number of features

`:param plot:` bool, if plt.show()

`:param save:` bool, if save the picture

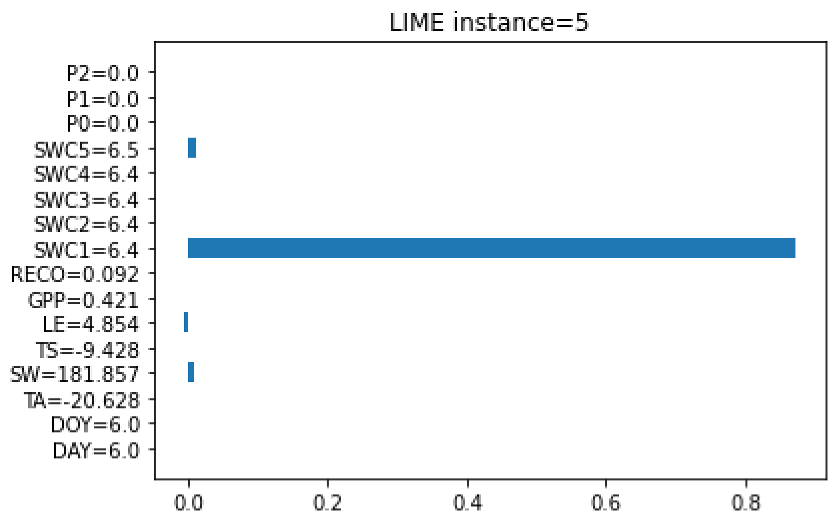
`:param save_path:` string, path of picture saved, default='lime.jpg'

`:return:` dataframe, lime values

For example,

```
from ExplainAI.explainers.lime.lime_xai import lime_xai
lime_res=lime_xai(m=m,x=x,y_ob=y_ob,instance=5,n=10000,num_bins=25)
print(lime_res)
```

```
#print(lime)
     feature  lime_var       value      ystick
DAY      DAY  0.000903    6.000000     DAY=6.0
DOY      DOY  0.000000    6.000000     DOY=6.0
TA        TA  0.000000  -20.628000  TA=-20.628
SW        SW  0.009448  181.857000  SW=181.857
TS        TS  0.000000   -9.428000   TS=-9.428
LE        LE -0.004269    4.853500    LE=4.854
GPP      GPP -0.000000    0.420799   GPP=0.421
RECO    RECO -0.000000    0.091506  RECO=0.092
SWC1    SWC1  0.871304    6.400000    SWC1=6.4
SWC2    SWC2  0.000000    6.400000    SWC2=6.4
SWC3    SWC3  0.000000    6.400000    SWC3=6.4
SWC4    SWC4  0.000000    6.400000    SWC4=6.4
SWC5    SWC5  0.011949    6.500000    SWC5=6.5
P0        P0  0.000000    0.000000      P0=0.0
P1        P1  0.000000    0.000000      P1=0.0
P2        P2  0.000000    0.000000      P2=0.0
```



Contributing

ExplainAI uses MIT license; contributions are welcome!

- Source code: https://github.com/HuangFeini/ExplainAI.git

ExplainAI supports Python 3.6+ .

References

1. Apley, D. W., and Zhu, J.: Visualizing the effects of predictor variables in black box supervised learning models. arXiv.org. https://arxiv.org/abs/1612.08468, 2019.
2. Breiman, L.: Classification and regression based on a forest of trees using random inputs, Mach. Learn., 45(1), 5–32, doi:10.1023/a:1010933404324, 2001.
3. Friedman, J. H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), doi:10.1214/aos/1013203451, 2001.
4. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv., 51(5), 1–42, doi:10.1145/3236009, 2019.
5. Štrumbelj, E., and Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. Knowl. Inf. Syst., 41(3), 647–665, doi:10.1007/s10115-013-0679-x, 2013.
6. Shapley, L.S., and Roth, A.E.: The Shapley value: essays in honor of Lloyd S. Shapley. Cambridge University Press. https://www.amazon.com/Shapley-Value-Essays-Honor-Lloyd-ebook/dp/B00IE6MSSY, 1988.

Citation

please cite the following for the usage of ExplainAI toolbox.

Copyright licence

Changelog

In this version, you might have some problems as follows. And you can try the sulotion to fix that.

1. if in the linux, 'display' issue still exists.

   import matplotlib.pyplot as plt

   -->plt.switch_backend('agg')

2. About the sklearn version.

   from sklearn.metrics import check_scoring

   --> vi sklearn/metrics/**init**.py

   -->from scorer import check_scoring

   -->all=['check_scoring']