ExplainAI

by Feini Huang, Wei Shuangguan, Yongkun Zhang

Contact: huangfn3@mail2.sysu.edu.cn

Overview

Installation

ExplainAI works in Python 2.7 and Python 3.4+. Currently it requires scikit-learn 1.14+. You can install ExplainAI using pip:

```
pip install ExplainAI
```

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

- pandas
- numpy
- lime
- six
- sklearn
- scipy
- seaborn
- shap
- datetime
- matplotlib

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 nd 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 design of ExplainAI obeys OOP(object-oriented programming). 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.

```
#1.Prediction
#m:trained model, a sklearn object
#d:input data, a pandas dataframe
#r2:r-squared precision, float
#da:features matrix, a pandas dataframe
m,d,r2 = randomforest()
da = d.drop("SWC", axis=1)

#2.PDP interpretation
#pdp_obj:PDP object
#TS:a feature of interest
pdp_obj=pdp.pdp_isolate(model=m, dataset=da, model_features=da.columns, feature="TS")
#2.1.Plot
fig, axes =pdp.pdp_plot(pdp_obj,"TS")
plt.show()
#2.2.Dataframe
df=pdp_obj.count_data
df.to_csv("df.csv")

import pandas as pd
# ----1.input data, you can use your dataset if you change the file path.
# file="./flx_data/dataset.csv"
# # d=pd.read_csv(file)
# # print(d)
# here, we use example dataset (after data processing).
from flx_data.input import input_dataset
d=input_dataset(flag=0)
# print(d)

# ----2.get training set and testing set
from data_processing.split_data import split_data
xtr,ytr,xte,yte=split_data(d,target="SWC").split()
# print(xtr)


# ----3.modelling with machine learning
from model.make_model import make_model
m,res,y_predict=make_model(modeltype='GradientBoosting',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
```

```
print(res)

# ----4.check the prediction and observation distribution
from 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"])

# plt.show()

# ----5.permutation inmportance
from utils import get_x,get_features
x=get_x(d,target="SWC")
f=get_features(x)
from explainers.pi.pi import pi_trans,pi_plot
p = pi_trans(model=m, feature_names=list(da.columns), preserve=False)

# ----6.plot-based or text-based interpretation
print(p)
pi_plot(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 (http:....) 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 is available at http: \fluxnet.org\data\fluxnet2015-dataset\fullset-data-product.

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

Filename	Content	
FLX_CN-Ha2_FLUXNET2015_FULLSET_DD_2003-2005_1-4.csv	Raw site data downloaded from FLUXNET	data=input_dataset(flag=2)
dataset_process.csv	Data after entire data processing	data=input_dataset(flag=1)
dataset.csv	Data after data processing and contrived work	data=input_dataset(flag=0)

```

```

Data processing

Since the FLUXNET raw data can not be used directly used in modeling, the data processing offers a feasible way to process the raw data.

Certainly, if the users have other time-relating and lagged-relating variables, the functions can be modified. And the dataset can be replaced.

Read data and add time-relating variables

The original FLUXNET data with time series only has time-relating variable "TIMESTAMP", whose formation is year%month%day%. It can not be used as a time-series variable.

Via the time_add function, the DAY (day sequence of whole time) and DOY (day of year) are added.

```
from data_processing.add_variables import time_add
file='.\\flx_data\\FLX_CN-Ha2_FLUXNET2015_FULLSET_DD_2003-2005_1-4.csv'
data=pd.read_csv(file,header=0)
new_data=time_add(data)
```

Add lagged-relating variables

In this example, the soil moisture has time "memory" and the lagged precipitation also has impact on soil moisture prediction, the 1 to 7 days lagged values of these two variables are added in the dataset.

```
from data_processing.add_variables import lag_add
new_data=lag_add(data,sm_lag=7,p_lag=7)
#sm_lag and p_lag are the days of lagged soil moisture and precipitation. Defaults are 7.
```

Data cleaning and feature selection

data_cleaning() offers several data cleaning functions:

- Eliminate the observation without target values
- Eliminate irrelevant records in FLUXNET, like percentiles, quality index, RANDUNC, se, sd...
- Eliminate the features with too many (30%) Nan.

```
from data_processing.data_cleaning import data_cleaning
c1=data_cleaning(data)
d1=c1.elim_SM_nan()

c2=data_cleaning(d1)
d2=c2.drop_ir()

c3=data_cleaning(d2)
d3=c3.drop_nan_feature()
```

feature_selection() provides sequential backward selection (SBS) which based on random forest.

```
from data_processing.feature_selection import feature_selection
fs=feature_selection(data=data,target="SWC_F_MDS_1")
new_data=fs.sbs_rf(n_estimators=100)
```

data_processing_main() integrates data cleaning and feature selection.

```
from data_processing.data_processing_main import data_processing_main
#drop_ir: eliminate data of irrelevant records in FLUXNET,like percentiles, quality index,
RANDUNC, se, sd...
#drop_nan_feature:Eliminate the features with too many(30%) Nan.
#part:part of split_data
#n_estimator:sequential backward selection using random forest, n_estimator of random
forest
#sbs:whether use sbs
d=data_processing_main(data=data,
                      time_add=True,
                       lag_add=True,
                      elim_SM_nan=True, #eliminate data of SM Nan
                      drop_ir=True,
                      drop_nan_feature=True,
                      part=0.7,
                      n_estimator=10,
                      sbs=True)
dd,ss=d.total()
dd.to_csv("dd.csv")
#dd is new_dataset after data processing
ss.to_csv('ss.csv')
#ss is sbs result
```

Split dataset

split_data offers a way to split dataset into training set and testing set, according to the time-sequence (using data of time-ahead to predict feature data).

```
from data_processing.split_data import split_data
xtr,ytr,xte,yte=split_data(data,target="SWC",part=0.7).split()
#Or validating set is required.
xtr,ytr,xvl,yvl,xte,yte=split_data(data,target="SWC",part3=[0.7,0.2,0.1]).split3()
```

Black-box machine learning

For this version, sklean models are available.

```
from model.make_model import make_model
model_list = ['DecisionTree', 'Linear', 'KNeighbors',
                  'RandomForest', 'AdaBoost',
                  'GradientBoosting', 'Bagging',
                  'BayesianRidge', 'SVR']
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
#model:trained model object
#y_predction:series, precdiction of testing set
```

Preview

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

```
# from preview import info_plots
# import matplotlib.pyplot as plt
from 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"])
#
plt.show()
```

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.

```
from model.randomforest_gv import randomforest
from explainers.mfi.mfi import mse_feature_importance,mse_feature_importance_plot

m,res,y_predict=make_model(modeltype='RandomForest',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
mfii=mse_feature_importance(model=m, data=data, preserve=True)
print(mfii)
mse_feature_importance_plot(mfii)
```

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.

pi_trans() offers an approach of PI storage in a dataframe.

pi_plot() supports plot of ranking features.

```
#get feature names list
from utils import get_x,get_features
f=get_features(x)

from explainers.pi.pi import pi_trans,pi_plot
p = pi_trans(model=m, feature_names=f, preserve=True)
pi_plot(p)
```

Partial dependence plot

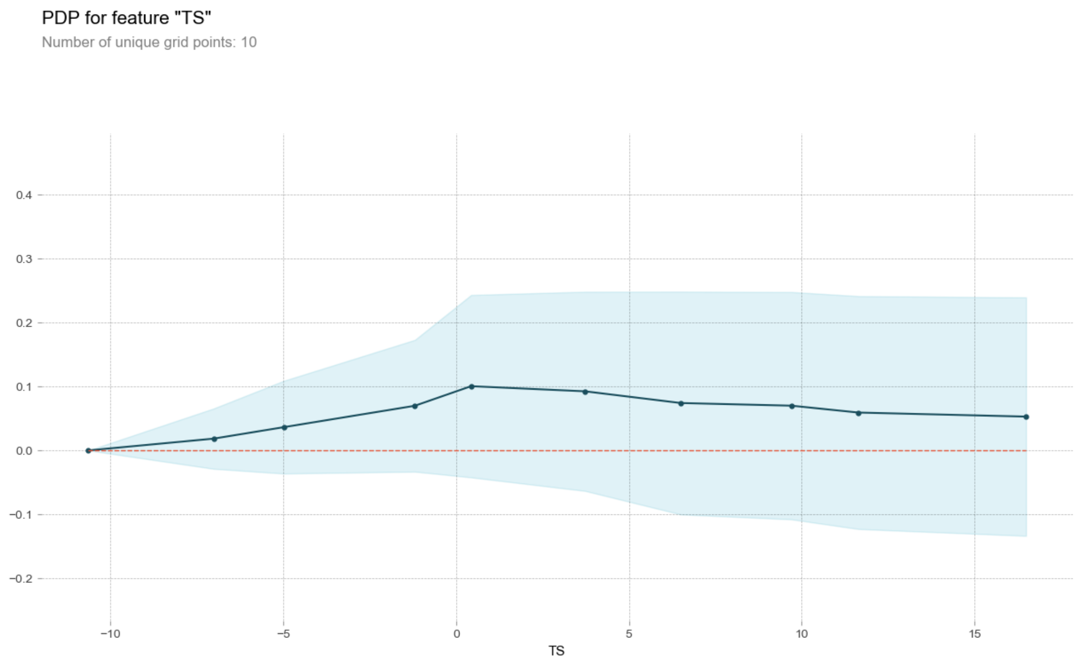
The PDP demonstrates the relationships between the features and predicted variable (Friedman, 2001). The PDP for regression is defined as:

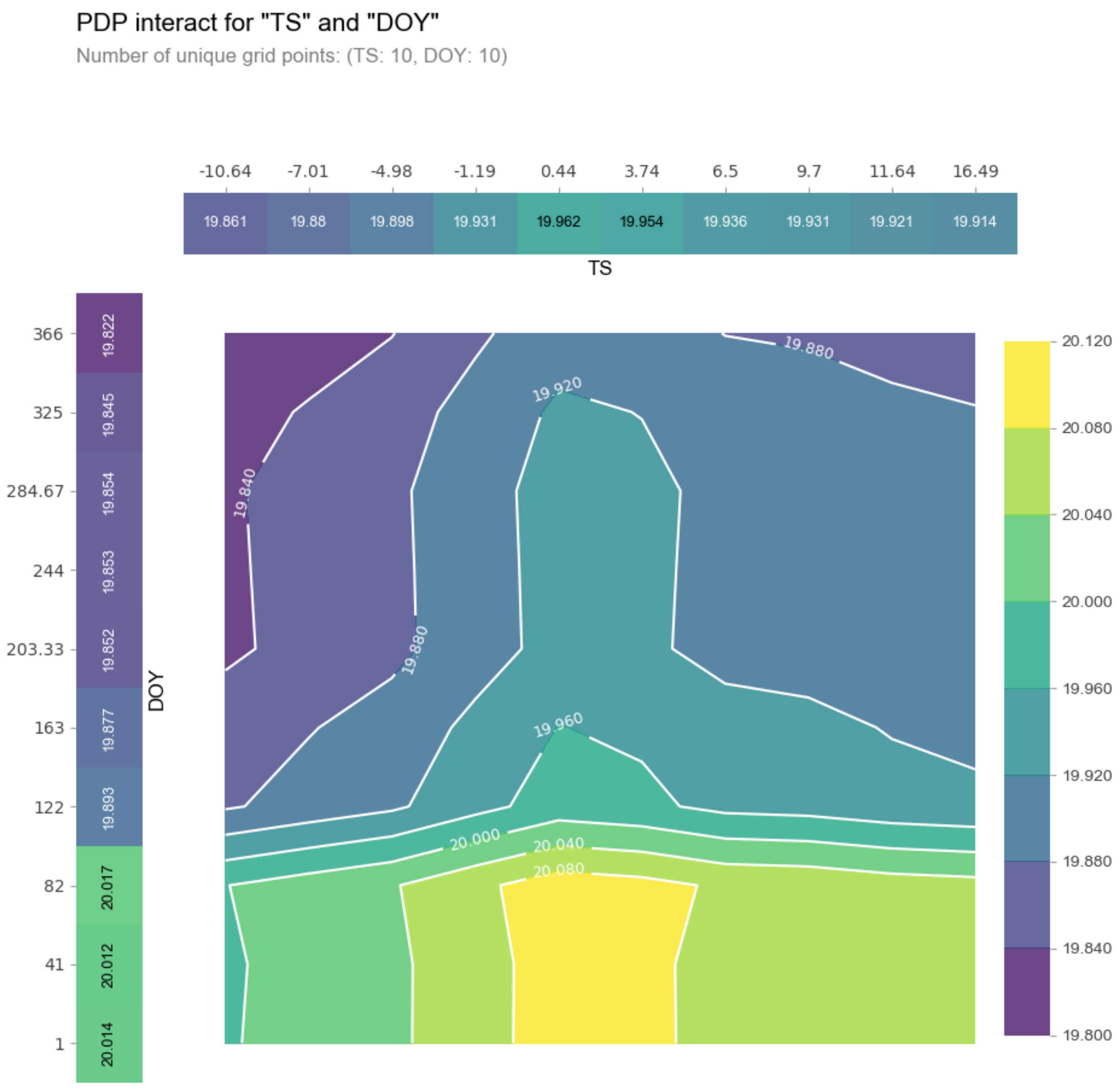
$$p(x(s,j))(x(s,j)) = 1/n \sum^{n} f[x(s,j), x_c(i)]$$

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.

```
from explainers.pdp import pdp
from utils import get_x,get_features
x=get_x(d,target="SWC") # feature matrix
f=get_features(x) # feature names list

#1.one-dimentional PDP object
pdp1=pdp.pdp_isolate(model=m,
                     dataset=x,
                     model_features=f,
                     feature="TS")
#2.PDP plot
fig3, axes =pdp.pdp_plot(pdp1,"TS")
plt.show()

#3.obtain PDP result as dataframe
print(pdp1.count_data)

#4.two-dimentional PDP object
pdp2=pdp.pdp_interact(model=m,
                      dataset=x,
                      model_features=f,
                      features=["TS","DOY"])
# 5.PDP plot
fig4, axes = pdp.pdp_interact_plot(
    pdp_interact_out=pdp2,
    feature_names=["TS","DOY"],
    plot_pdp='contour')
#6.obtain PDP result as dataframe
print(pdp2.pdp)
plt.show()
```

```
   x      xticklabels  count  count_norm
0  0  [-10.64, -7.01)    122    0.111314
1  1   [-7.01, -4.98)    122    0.111314
2  2   [-4.98, -1.19)    121    0.110401
3  3    [-1.19, 0.44)    122    0.111314
4  4     [0.44, 3.74)    122    0.111314
5  5      [3.74, 6.5)    121    0.110401
6  6       [6.5, 9.7)    122    0.111314
7  7     [9.7, 11.64)    122    0.111314
8  8   [11.64, 16.49]    122    0.111314
```





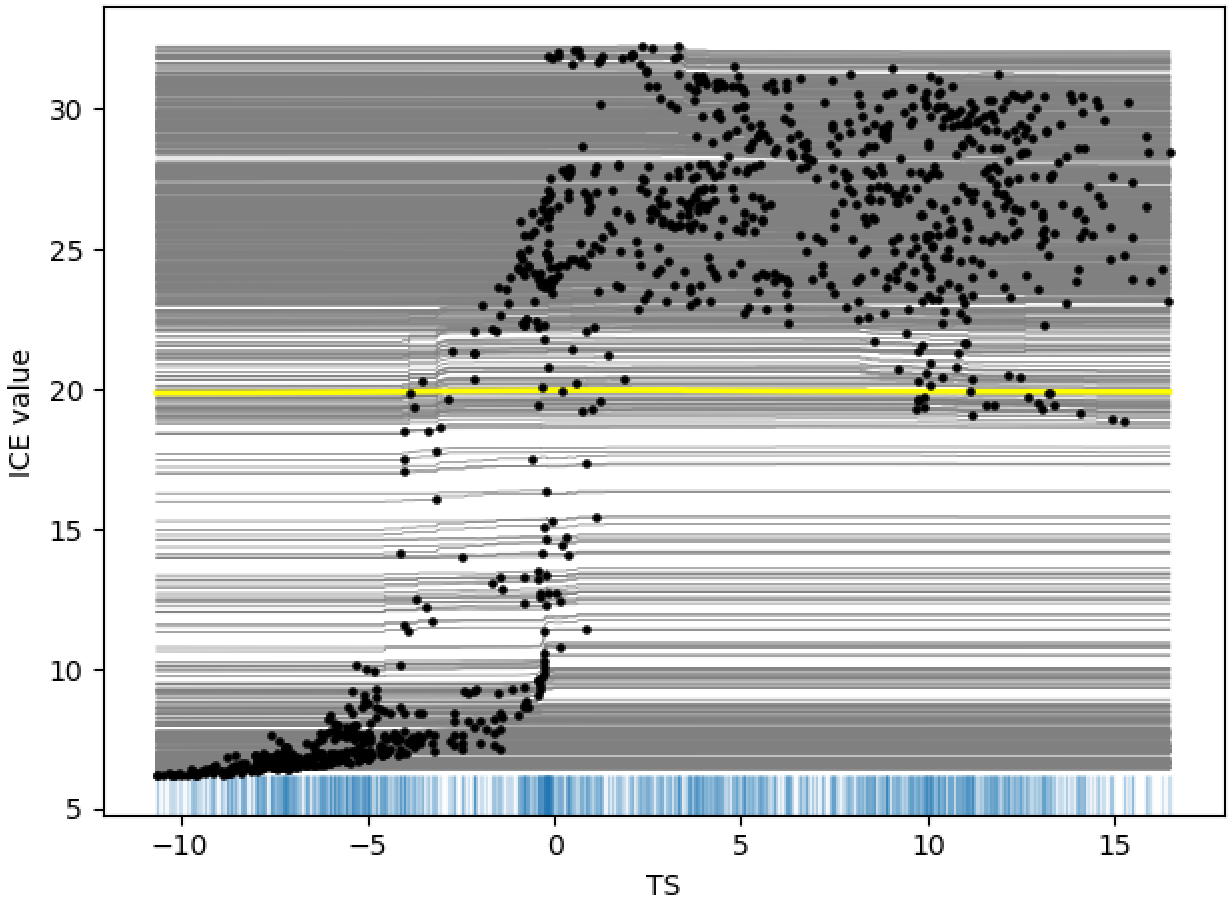
Individual conditional expectation

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.

```
from explainers.ice import ice
ice_obj=ice.ice(data=x,column="TS",predict=m.predict)
icep=ice.ice_plot(ice_obj,
                        column="TS",
                        plot_points=True,
                        color_by=None,
                        plot_pdp=True)
plt.show()
```



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.

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

$$h = k_j(k)$$

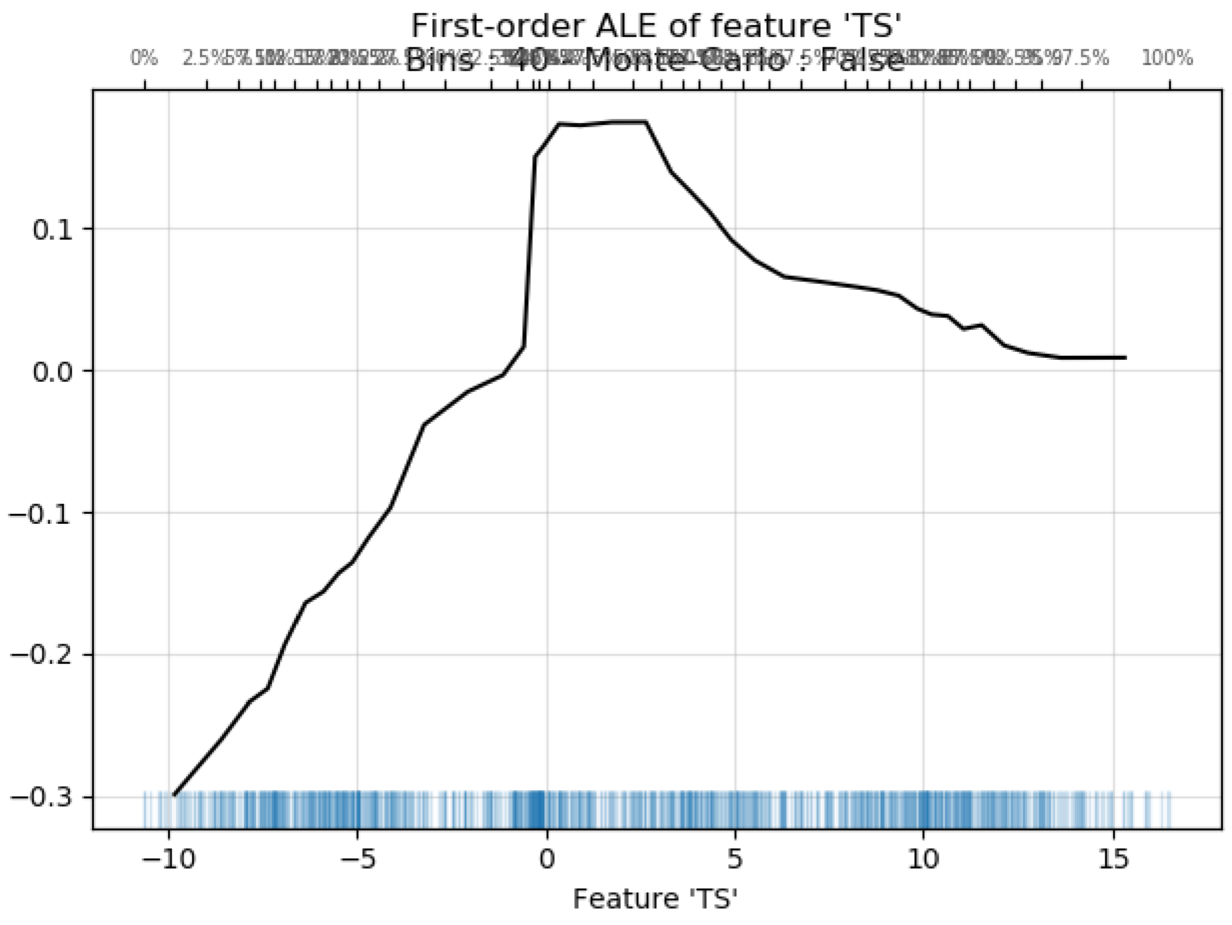
$$m = (i : x(i,j) \in N_j(k))$$

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

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

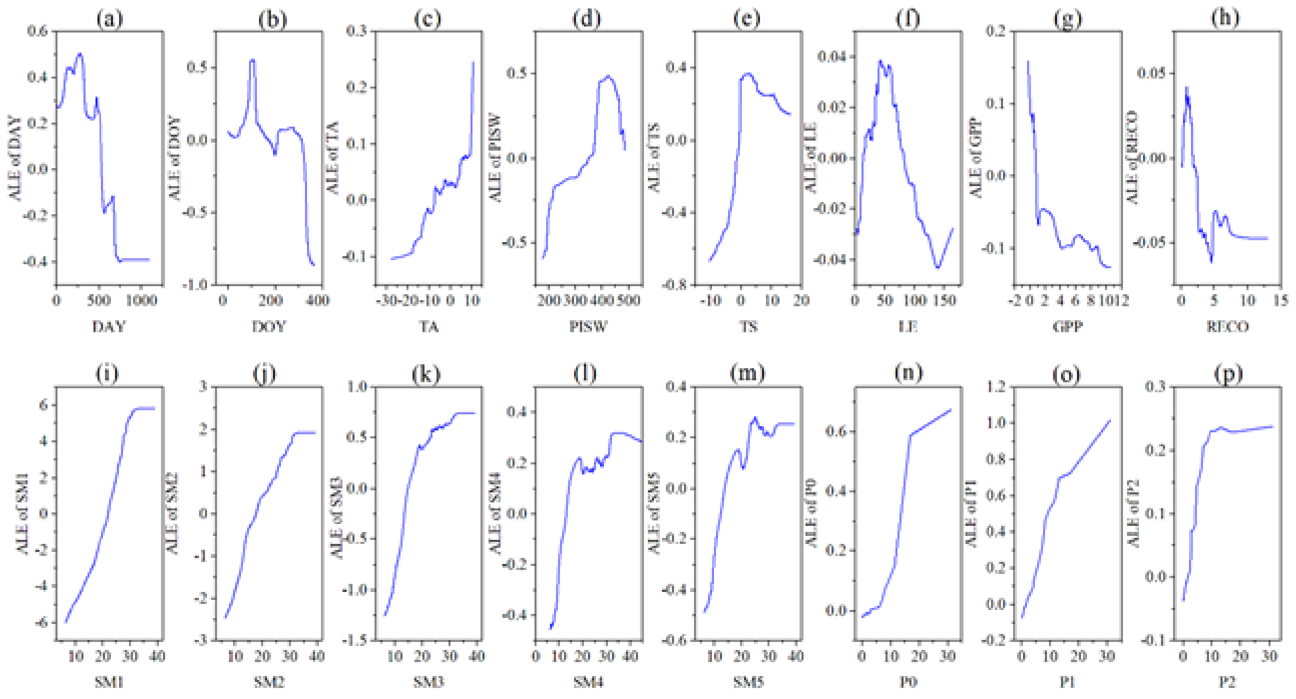
where fj donates the ALE values and it visualizes the main effect dependence of modelling on x_j, x(i,\j)=(x(i,l):l=1, …,d;l≠j), where the subscript \j means all feature but the j-th. Similarly, Nj (k)=(z(k-1,j),z(k,j);k=1,2,…,K) donates a sufficiently fine partition of the sample range of x(i,j) into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

```
from explainers.ale.ale import ale_plot
import matplotlib.pyplot as plt
from explainers.ale.ale_output import ale_output,ale_plot_total
#1.one-dimentional ale
ale_plot(model=m, train_set=x, features='TS',plot=True)

#2.two-dimentional ale
ale_plot(m, train_set=x, features=tuple(['TS', 'DOY']), plot=True, bins=40,
monte_carlo=False)

#3.one-dimentional ale plots of all features
#plot
ale_plot_total(model=m, data=x)
#data_frame in csv
ale_output(model=m, data=x,preserve=True)
```

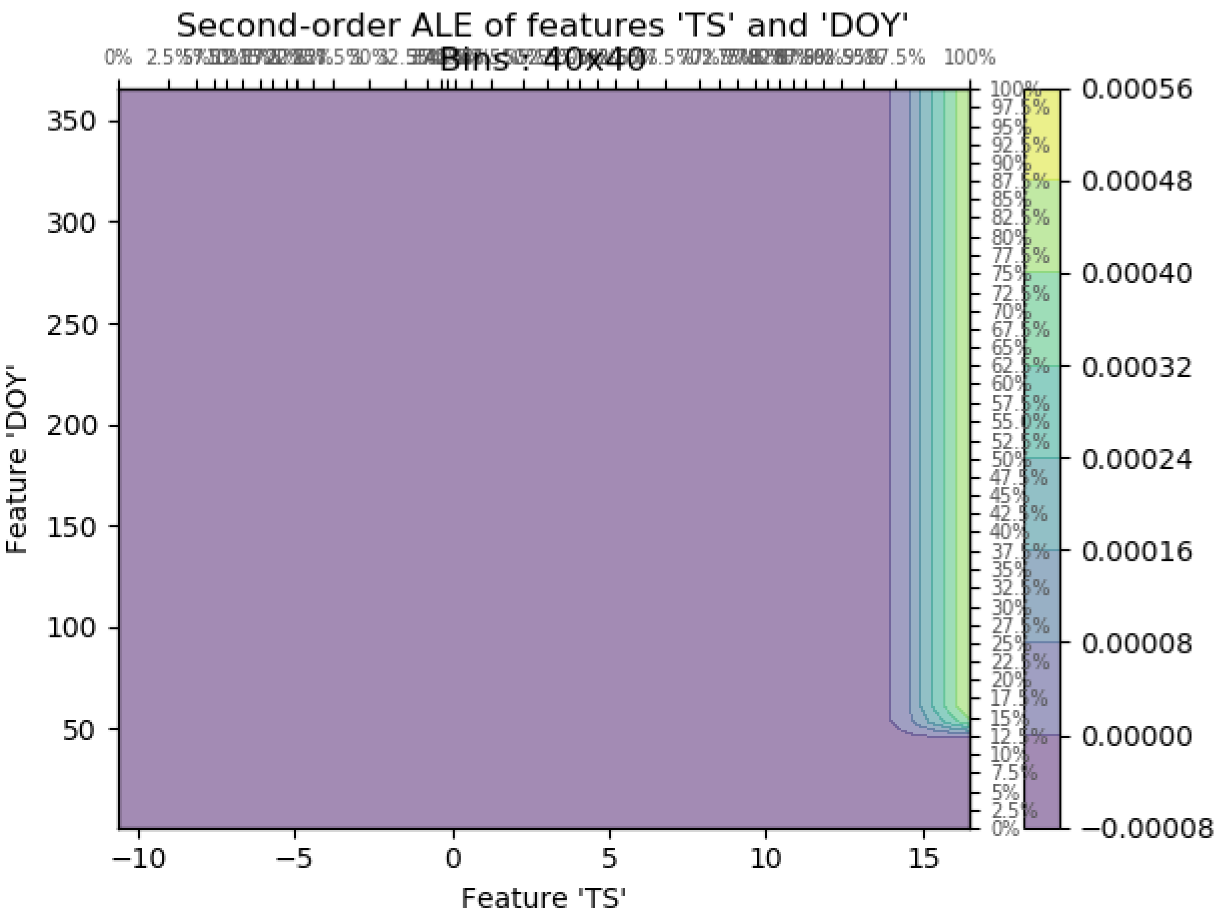


All one-dimensional ALE plots can be presented by ale_plot_total().

(C:\Users\Acer\AppData\Roaming\Typora\typora-user-images\1643785450077.png)



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.



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.

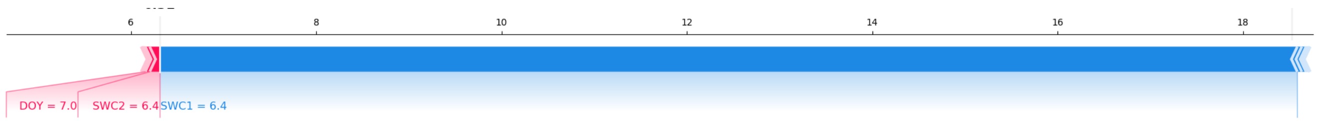
At first, the Shapley values function is treated as a vessel.

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

```
from model.randomforest_gv import randomforest
from explainers.shap_func.shap_func import shap_func
ss=shap_func(m,x)
ss.record_shap()
```

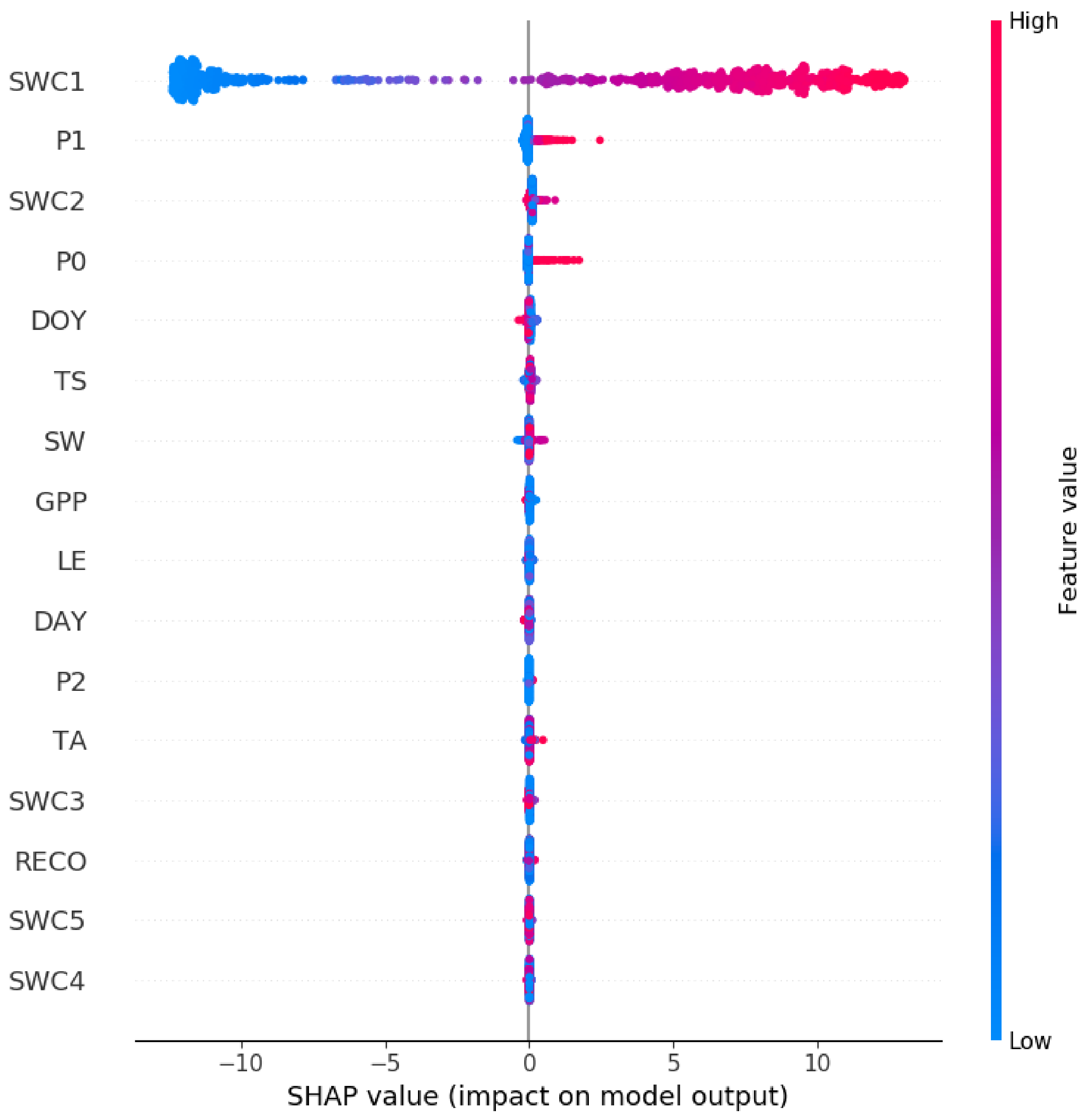
single_shap() offers shapley values for an individual instance.

```
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()
```

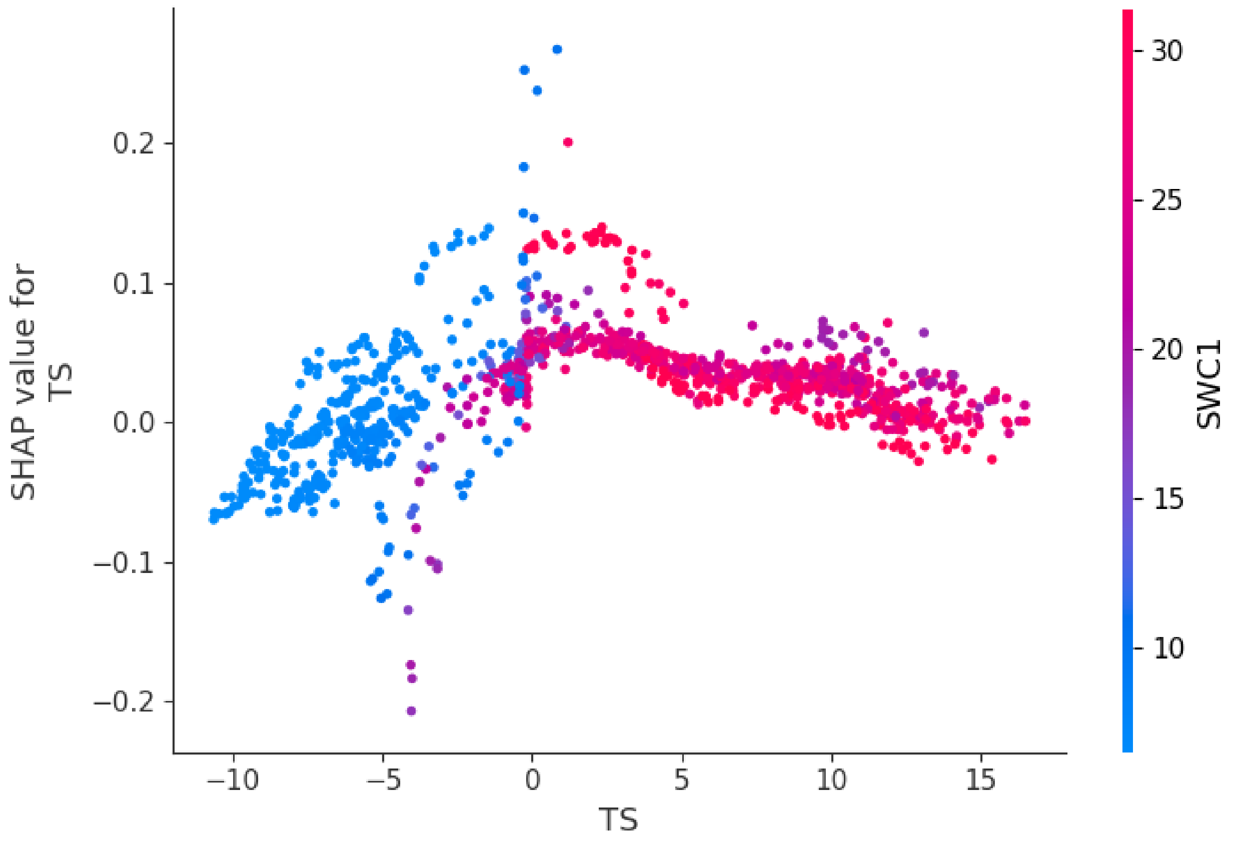


time_shap() provides Shapley values distribution with time series.

```
ss.time_shap()
```

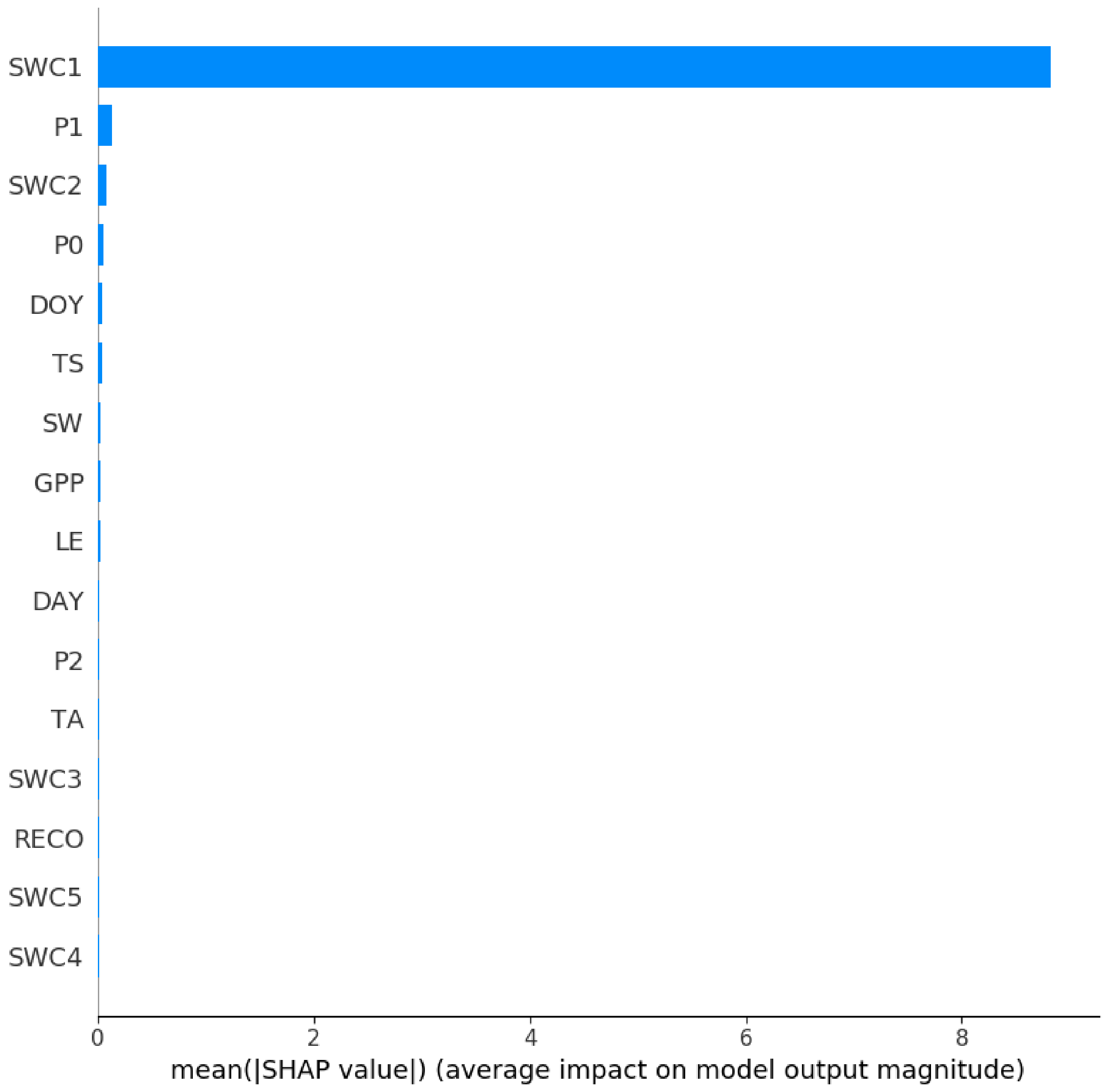
depend_shap() provides Shapley values distribution with TS variation.

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



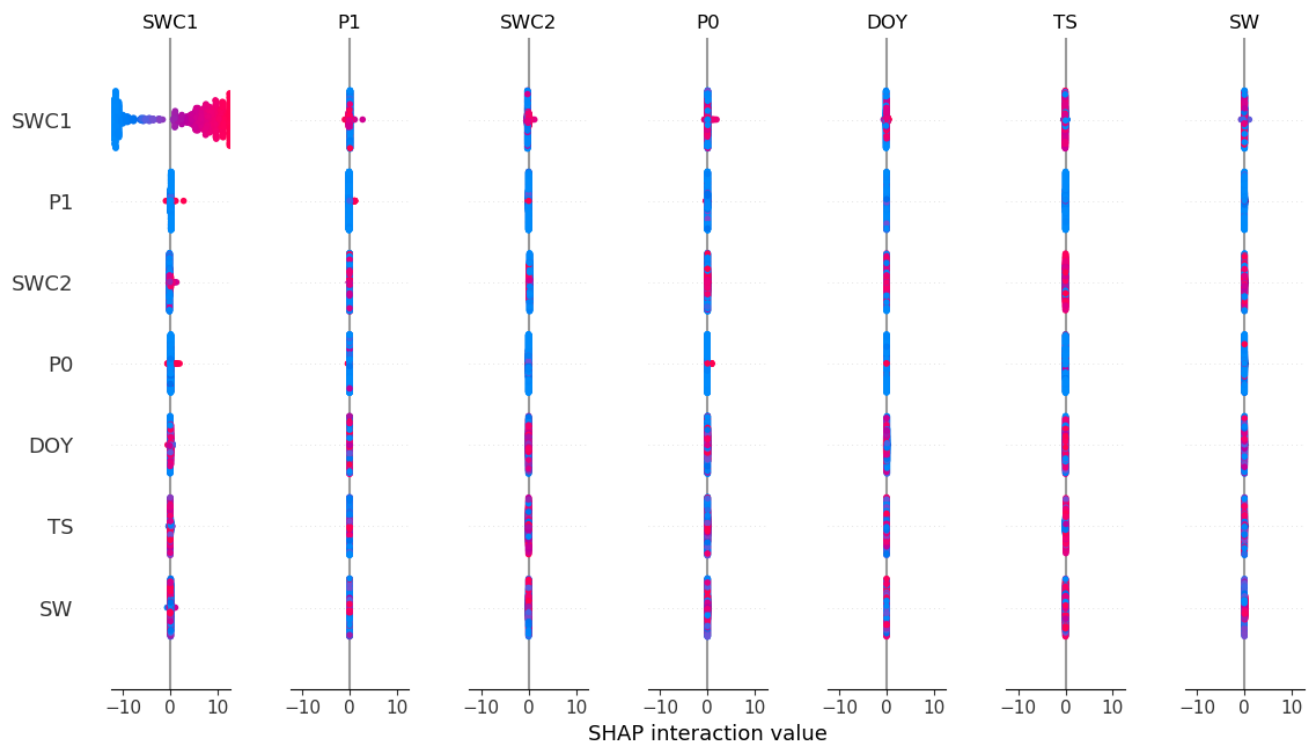
mean_shap() provides averaged Shapley values of features.

```
ss.mean_shap()
```



intera_shap() provides averaged Shapley values under two features' interaction.

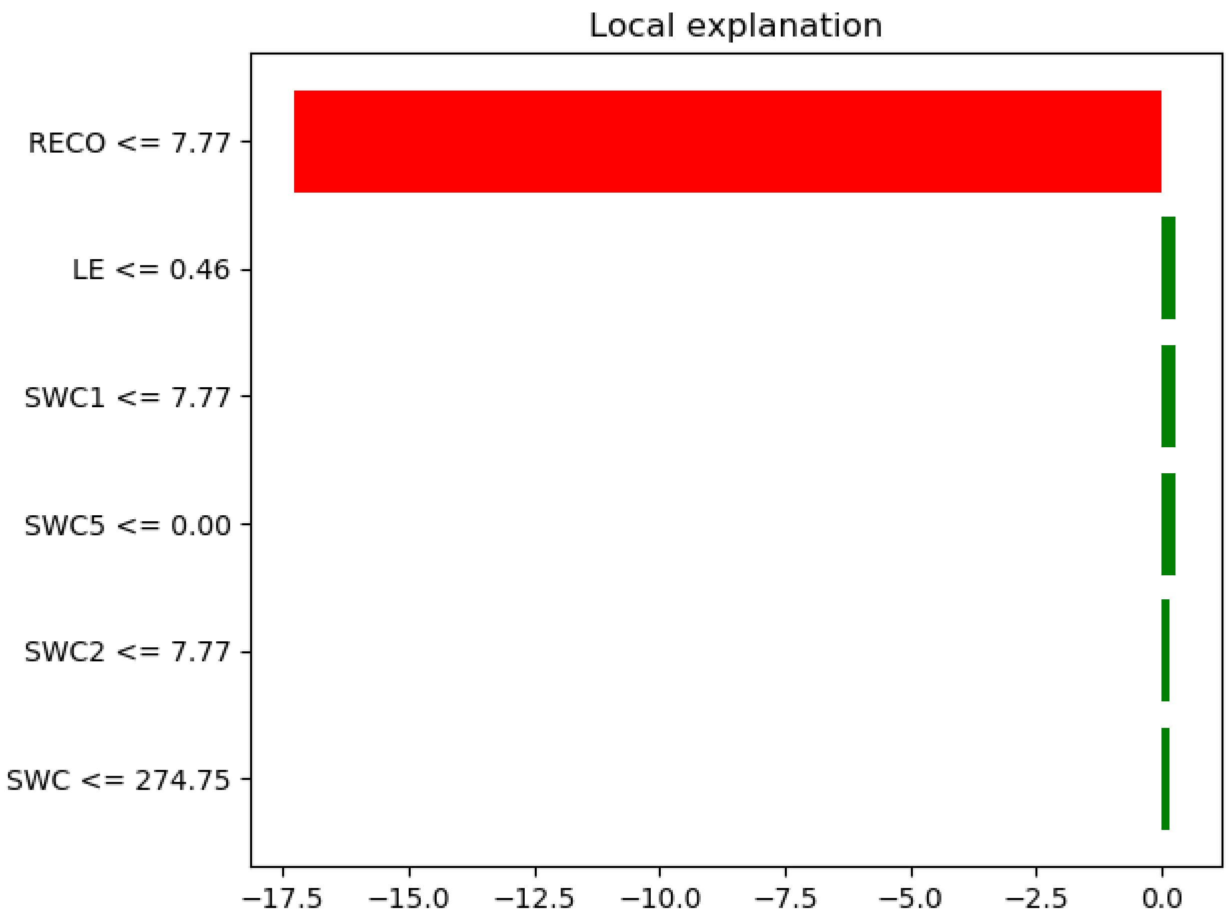
```
ss.intera_shap()
```



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.

```
from explainers.lime_func.lime_output import lime_func,lime_output
import numpy as np

exp = lime_func(model=m,
                train_data=np.array(x),
                feature_names=f,
                target_feature="TS",
                instance=x[0])#number 0 is the sequence of instance.
ins, out = lime_output(exp, plot=True)
```



Contributing

ExplainAI uses MIT license; contributions are welcome!

- Source code: <https://github.com/HuangFeini/xai

ExplainAI supports Python 2.7 and Python 3.4+ .

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.

Changelog