ExplainAI

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
```

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 packages:

- scikit-learn. Currently ExplainAI allows to explain predictions of scikit-learn regressors (at present, random forest only), show feature importances and feature effects.
- Post-hoc interpretation. Including partial dependence plot (PDP), mean squared error (MSE)-based feature importance (MFI), permutation feature importance, accumulated local effect (ALE), individual

conditional expectation (ICE), local nd Shapley values, ExplainAI can integrate those present post-hoc methods.
- Data preview. You can get observation and prediction distribution with feature or feature interaction variation, which are displayed in a figure of console.
- At this version, tabular data formation can be supported. And 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.

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 washing
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. interpretation. The trained model, input data matrix as input, the interpretation methods can be *objectified.
5. display the results of interpretation (plot or text).

There are two main ways to look at a classification or a regression model:

1. inspect model parameters 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 provide the supports.

   For (2), ICE, Shapley values and LIME provide the supports.

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")
```

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 a 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 (paper) try to explain a black-box classifier through a locally-fit simple, interpretable classifier. It means that with each additional supported “simple” regressor algorithms like LIME are getting more options automatically.

Tutorials

Dataset

All dataset used in this case is in the storage of flx_data.

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

Filename	Content	
D:\codes\xai\flx_data\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.

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 present a time-series variable.

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

```
from feature_selection.feature_selection import time_add
file='D:\\codes\\xai\\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

Due to in this case, the soil moisture has time memory and the lagged precipitation also has impact on soil moisture prediction, the lagged-relating variables should be added in the dataset.

```
from feature_selection.feature_selection import lag_add
file='D:\\codes\\xai\\flx_data\\FLX_CN-Ha2_FLUXNET2015_FULLSET_DD_2003-2005_1-4.csv'
data=pd.read_csv(file,header=0)
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.
```

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 feature_selection.feature_selection import split_data
s=split_data(data,part=0.7,part3=[0.7,0.2,0.1])
train,test=s.split()
#Or validating set is required.
train, valid, test=s.split3()
```

Data processing

data_processing_main() integrates data washing and feature selection.

```
from feature_selection.feature_selection import data_processing_main
file='D:\\codes\\xai\\flx_data\\FLX_CN-Ha2_FLUXNET2015_FULLSET_DD_2003-2005_1-4.csv'
data=pd.read_csv(file,header=0)
#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
```

Black-box machine learning

At this version, only random forest is available.

```
from model.randomforest_gv import randomforest
#re_feature_selection: boolean, if False, input data from dataset.csv (via contrived
work).if True, input data from feature_selection.
#prediction_record:boolean,if True, prediction saved. if False, prediction no saved.
model_best,data,r2=randomforest(prediction_record=False,re_feature_selection=False)
#model_best:RandomForestRegressor object.
#data:dataframe, same as input data.
#r2:float, prediction precision of testing set.
```

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.

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

m, d, r2 = randomforest(re_feature_selection=False)
mfii=mse_feature_importance(model=m, data=d, preserve=False)
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.

```
from model.randomforest_gv import randomforest
from explainers.pi.pi import pi_trans,pi_plot
m,d,r2 = randomforest()
da = d.drop("SWC", axis=1)
p=pi_trans(model=m,feature_names=list(da.columns),preserve=False)
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:

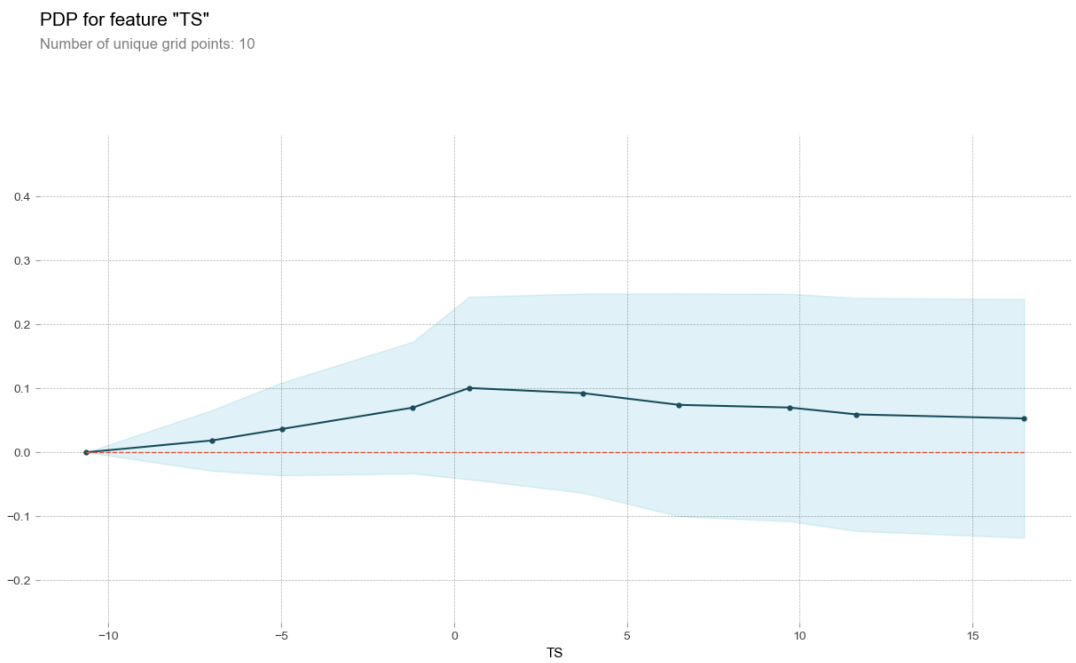
$$p\hat{}_(x_(s,j))(x_(s,j)) = 1/n \sum_(i = 1)^n ▒ 〖 f\hat{} (x_(s,j), x_c^((i)))〗$$

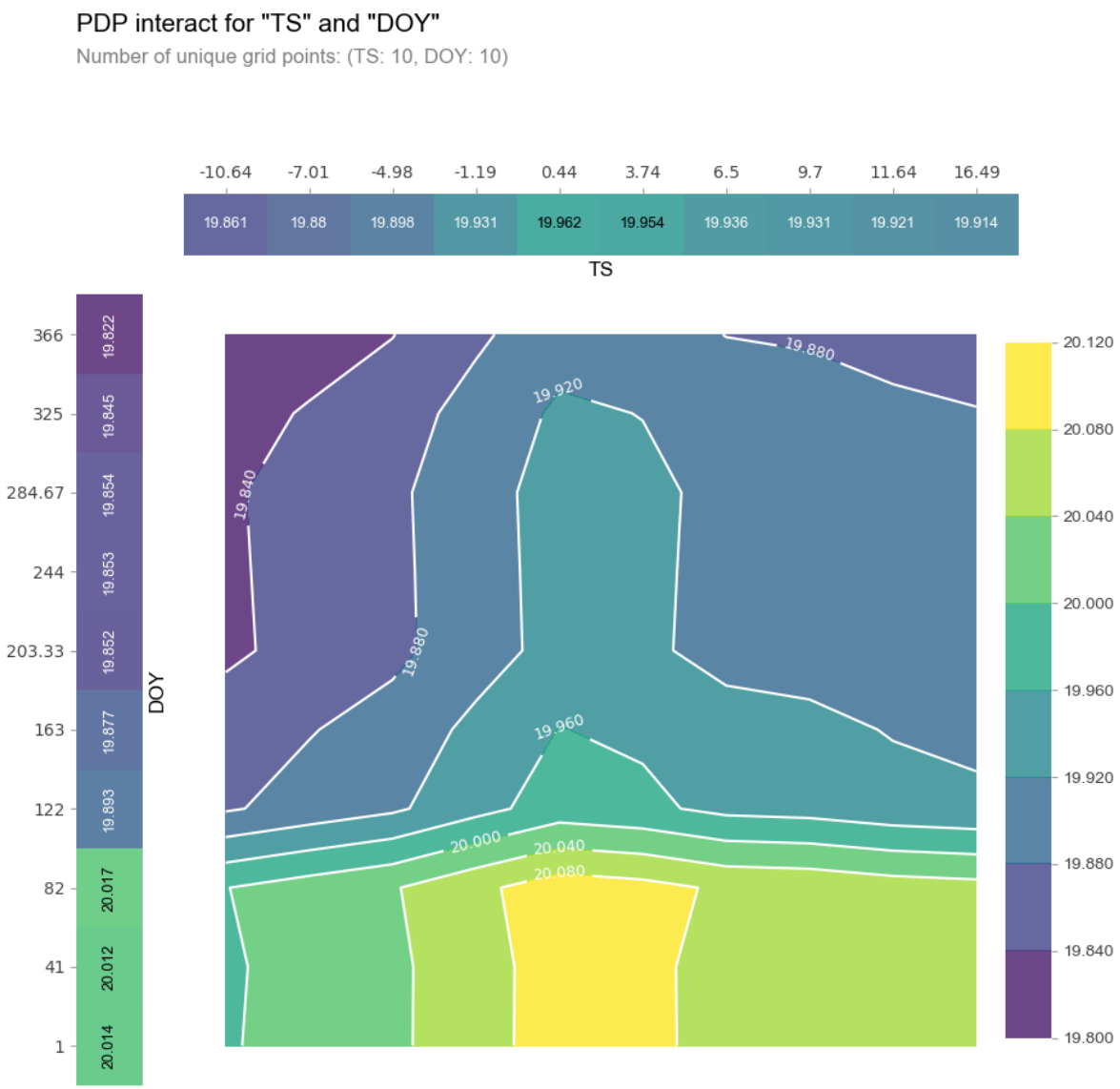
$$where x_(s,j) is the set of the feature of interest (as j - th feature) for which the partial dependence function should be plotted, p\hat{}_(x_(s,j)) is the partial dependence value of j - th feature, n is the number of elements in x_s, and x_c is subset of the ractual features values. PDP estimates the average marginal effect of predictors on the predicted SM, which can be a determined value in regression. Aiming at visualization of feature effects, the PDP can be conducted by the following steps (Fig. S2): a) the predictors P = x_1, x_2, \ldots, x_k, \ldots, x_p are divided into two subsets, including the feature of interest S = x_k and other features C = x_1, x_2, \ldots, x_(k-1), x_(k+1) \ldots, x_p. Wherein, the S subset involves possible values of x_k, which can be described as x_s = x_(s,1), x_(s,2), \ldots, x_(s,j), \ldots, x_(s,m).$$

```
import matplotlib.pyplot as plt
from model.randomforest_gv import randomforest
from explainers.pdp import pdp

#1.modeling
m,d,r2 = randomforest(re_feature_selection=False)
da = d.drop("SWC", axis=1)

#2.one-dimentional PDP object
pdp1=pdp.pdp_isolate(model=m,
                     dataset=da,
                     model_features=da.columns,
                     feature="TS")
#2.1.PDP plot
fig3, axes =pdp.pdp_plot(pdp1,"TS")
plt.show()
#2.2.obtain PDP result as dataframe
print(pdp1.count_data)

#3.two-dimentional PDP object
pdp2=pdp.pdp_interact(model=m,
                      dataset=da,
                      model_features=da.columns,
                      features=["TS","DOY"])
# 3.1.PDP plot
fig4, axes = pdp.pdp_interact_plot(
    pdp_interact_out=pdp2,
    feature_names=["TS","DOY"],
    plot_pdp='contour')
#3.2.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

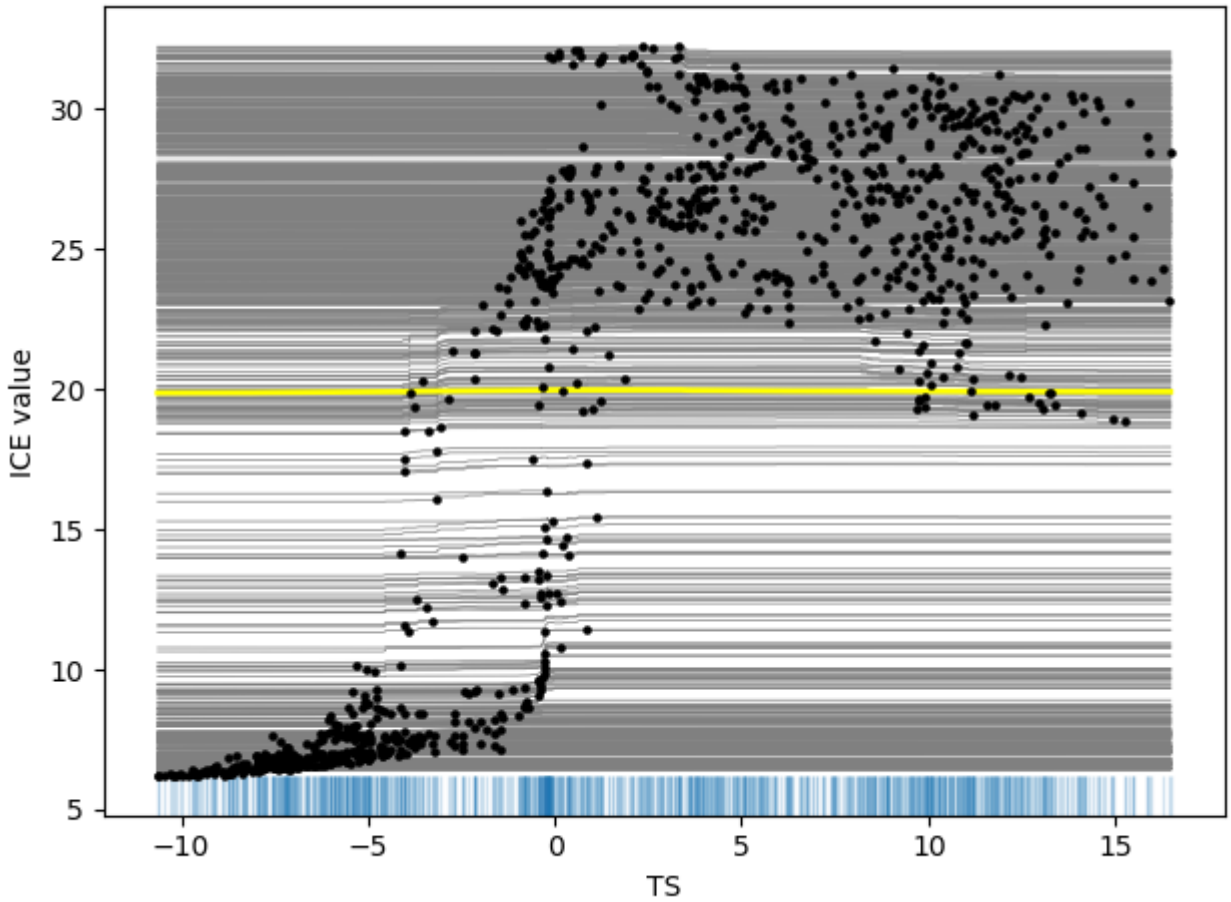
ICE was proposed by Goldstein et al. (2015). The ICE concept is given by:

$$\hat{p}_{(}x_{(}s,j))(x_{(}s,j)) = \hat{f}(x_{(}s,j),x_c)$$

$$< Empty\ Math\ Block >$$

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
from model.randomforest_gv import randomforest
import matplotlib.pyplot as plt
m,d,r2 = randomforest(re_feature_selection=False)
d = d.drop("SWC", axis=1)
ice_obj=ice.ice(data=d,column="TS",predict=m.predict)
icep=ice.ice_plot(ice_obj,
                  column="TS",
                  plot_points=True,
                  color_by=None,
                  plot_pdp=True)
plt.show()
```



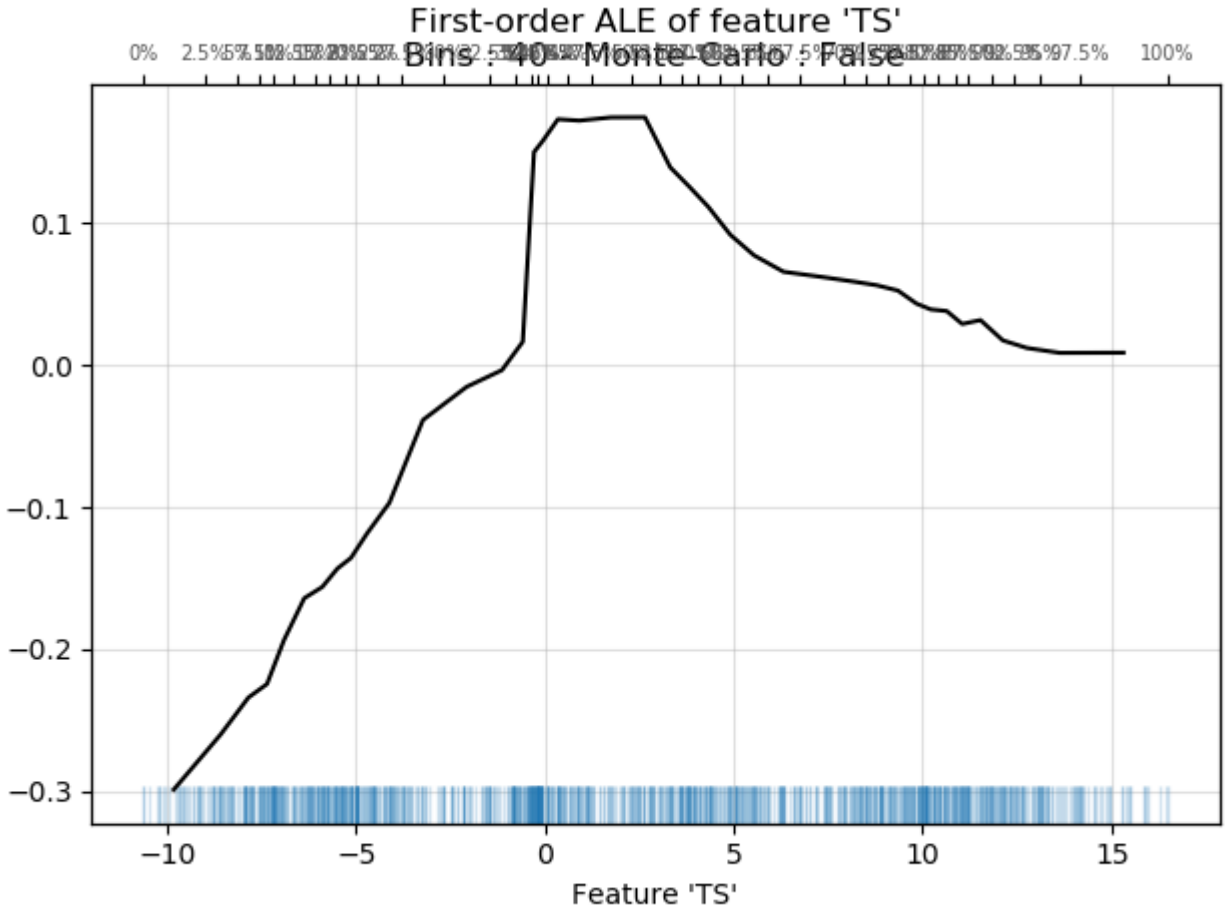
Accumulated Local Effect

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

$$(f_j)\hat{}(x) = \sum_{(} k = 1)^{(} k_j(x))\blacksquare \llbracket 1/(n_j(k)) \sum_{(} i : x_{(} i, j) \in N_j(k))\blacksquare \llbracket [f(z_{(} k, j), x_{(} i, \backslash j)\rrbracket \rrbracket) - f(z_{(} k-1, j), x_{(} i, \backslash j)) - c$$

where $(f_j)\hat{}$ denotes the ALE values and it is used to see the main effect dependence of model $\hat{f}$ on x_j. $x_{(} i, \backslash j) = (x_{(} i, l) : l = 1, \ldots, d; l \neq j)$, where the subscript $\backslash j$ means all features but the $j-th$. Similarly, $N_j(k) = (z_{(} k-1, j), z_{(} k, j)]; k = 1, 2, \ldots, K)$ denotes an efficiently fine partition of the sample range of $x_{(} i, j)$ into K intervals. And the constant c is chosen so that $1/n \sum_{(} i = 1)^n \llbracket (f_j)\hat{}(x)\rrbracket = 0$.

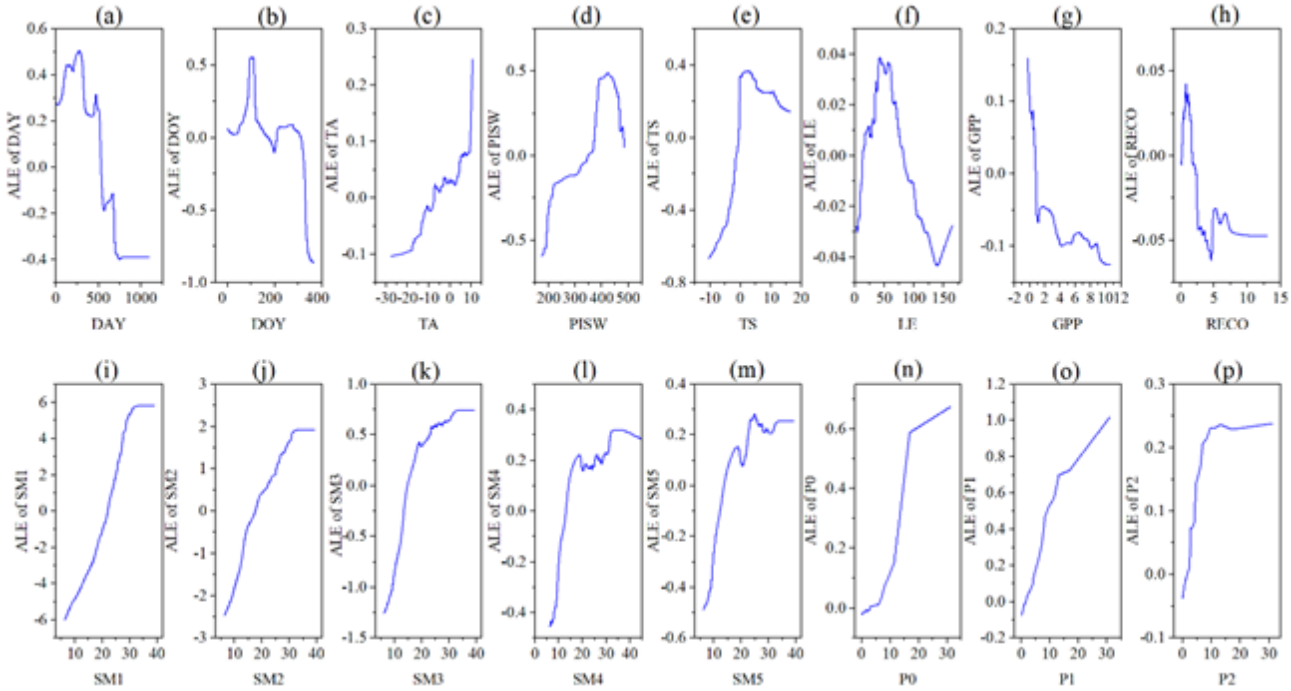
```
from explainers.ale.ale_output import ale_output,ale_plot_total
from explainers.ale.ale import ale_plot
from model.randomforest_gv import randomforest
import matplotlib.pyplot as plt

   best_model,data,r2=randomforest(re_feature_selection=False)
d = data.drop("SWC", axis=1)
ale_plot(best_model, train_set=d, features='TS', plot=False,bins=40, monte_carlo=False)
```



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

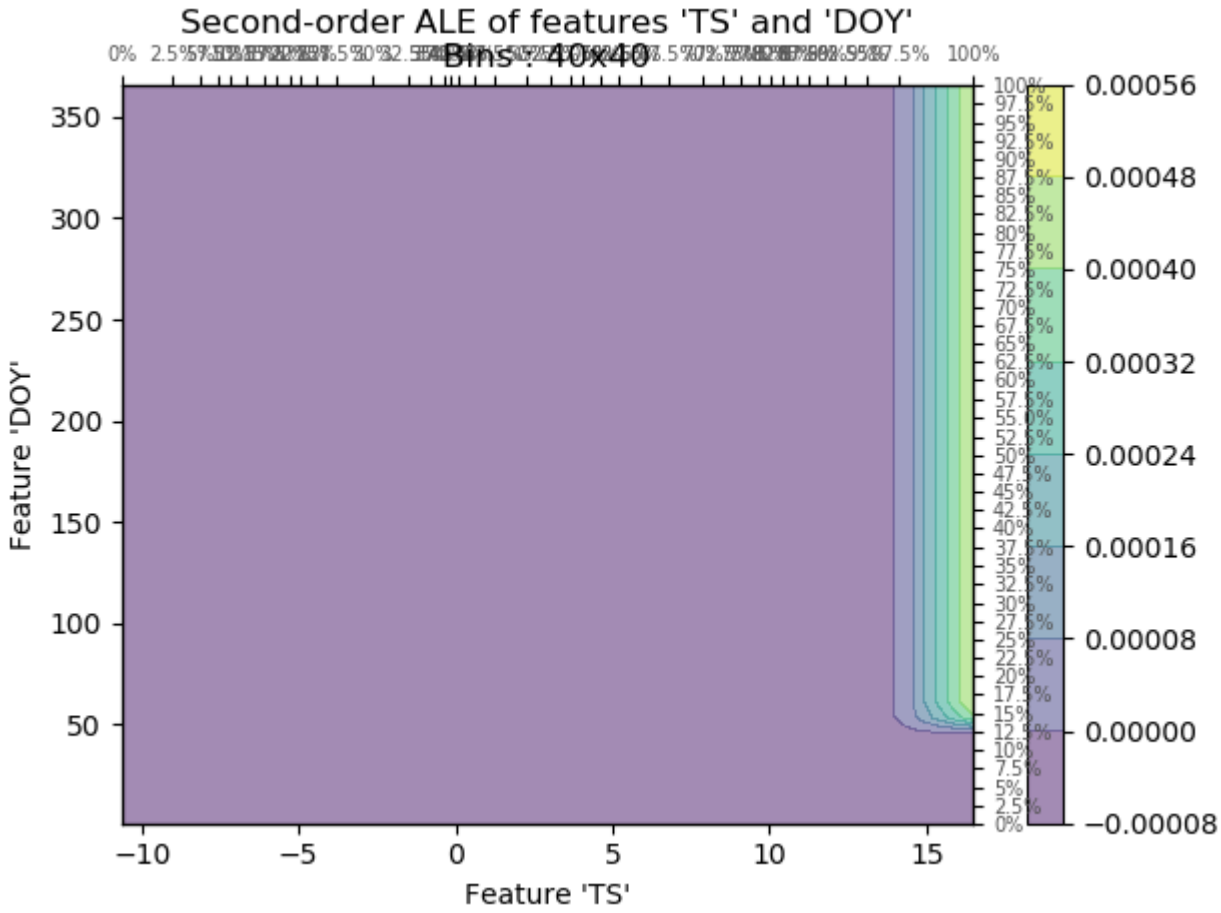
```
from explainers.ale.ale_output import ale_output,ale_plot_total
from explainers.ale.ale import ale_plot
from model.randomforest_gv import randomforest
import matplotlib.pyplot as plt
best_model,data,r2=randomforest(re_feature_selection=False)
d = data.drop("SWC", axis=1)
ale_plot_total(model=best_model,data=d)
```

(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.

```
from explainers.ale.ale_output import ale_output,ale_plot_total
from explainers.ale.ale import ale_plot
from model.randomforest_gv import randomforest
import matplotlib.pyplot as plt
best_model,data,r2=randomforest(re_feature_selection=False)
d = data.drop("SWC", axis=1)
ale_plot(best_model, train_set=d, features=tuple(['TS','DOY']), plot=False, bins=40,
monte_carlo=False)
plt.show()
```



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 SV method can be applied on any data subset or even a single instance. The Shapley values of a feature value is its contribution to the predicted result, weighted and summed over all possible feature value combinations:

$$\hat{\varphi}_{(}i,j)(val) = \sum_{(} S \subseteq x_{(}i,1,) \ldots, x_{(}i,p \setminus x_{(}i,j))) \frac{|S|!(p-|S|-1)!}{p!}[val(S \cup x_{(}i,j)) - val(S)]$$

where S is a subset of the features used in an alliance, x is the vector of feature value of interest of instance j, p denates 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.

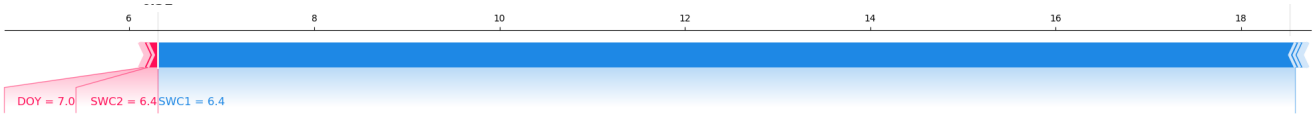
At first, the Shapley values function is treated as a vessel.

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
m,d,r2 = randomforest(re_feature_selection=False)
x = d.drop("SWC", axis=1)
y=d["SWC"]
ss=shap_func(m,x)
ss.record_shap()
```

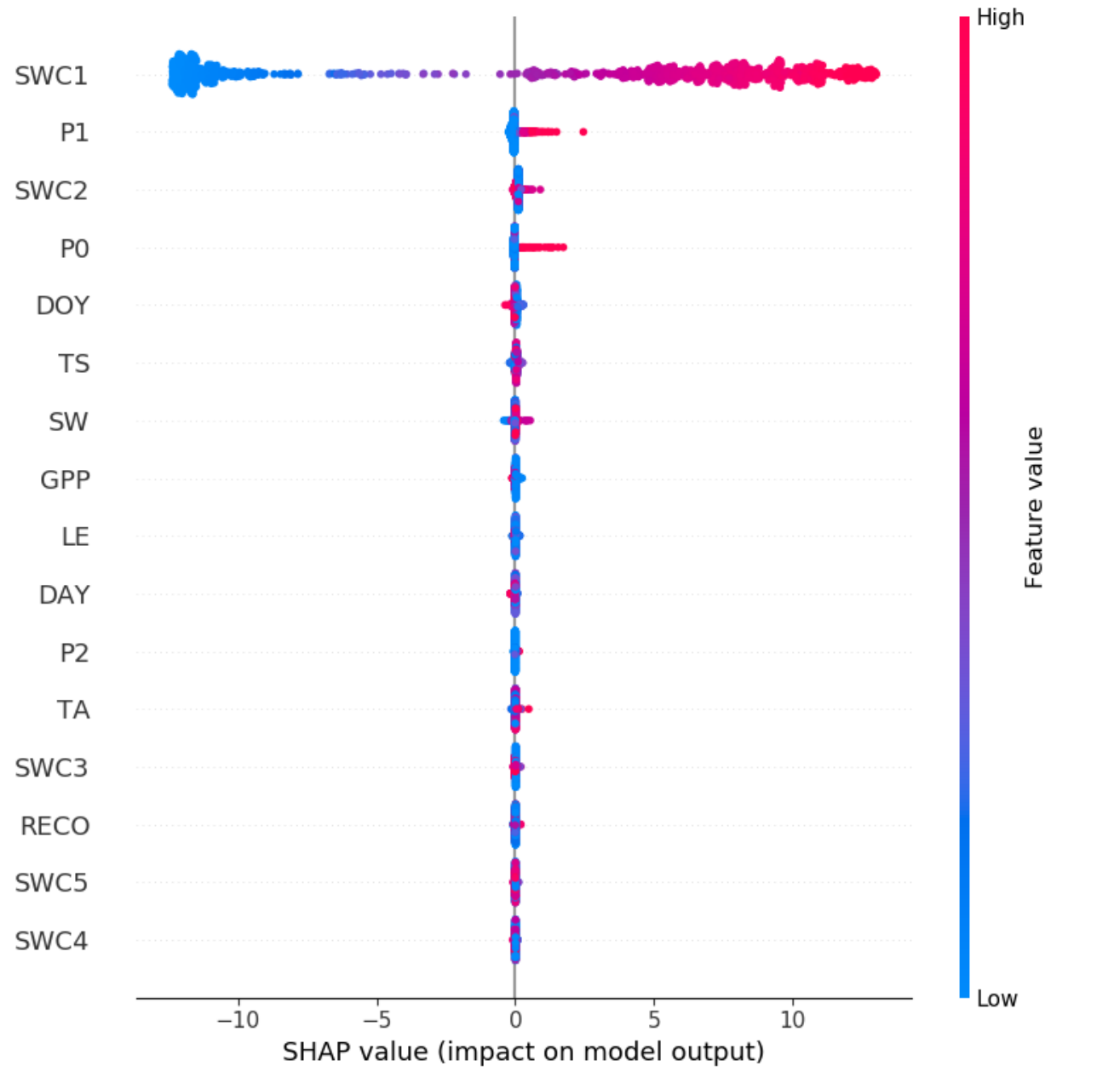
single_shap() offers shapley values for an individual instance.

```
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()
```

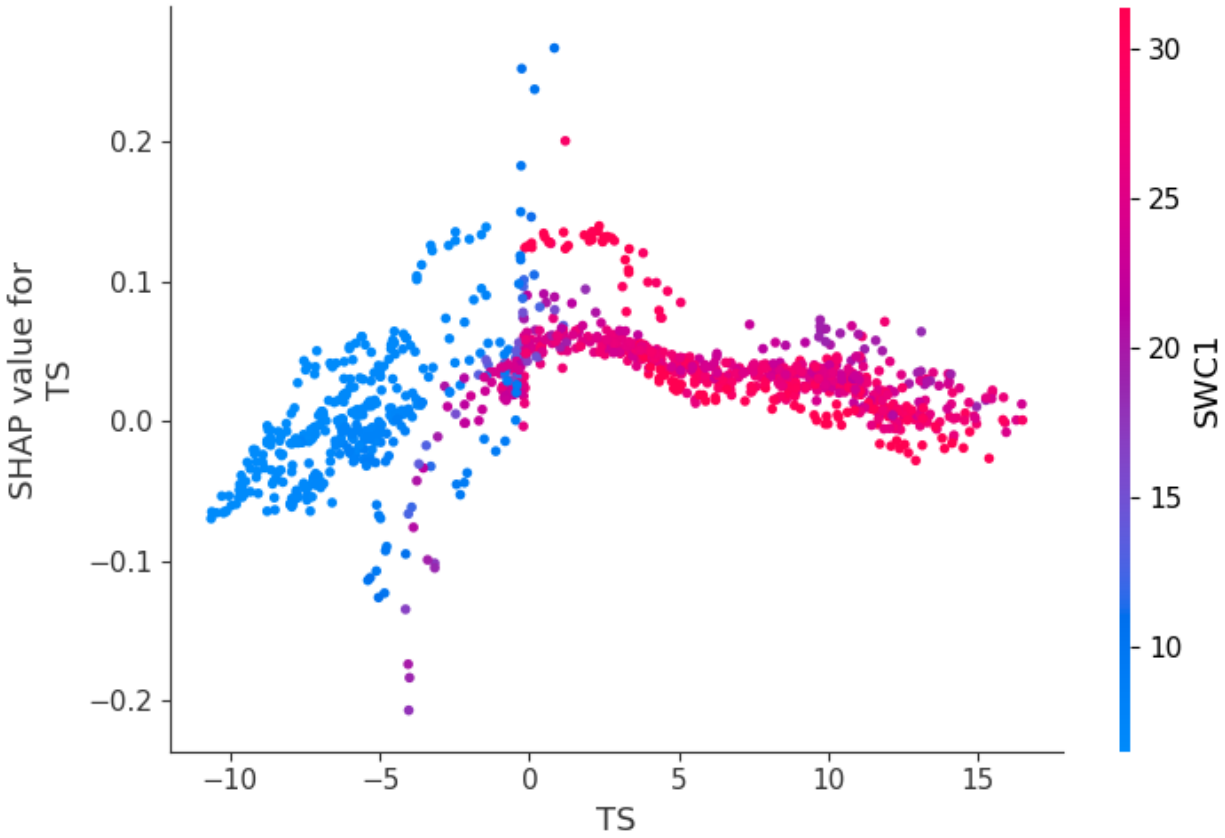


time_shap() provides Shapley values distribution with time series.

```
ss.time_shap()
```

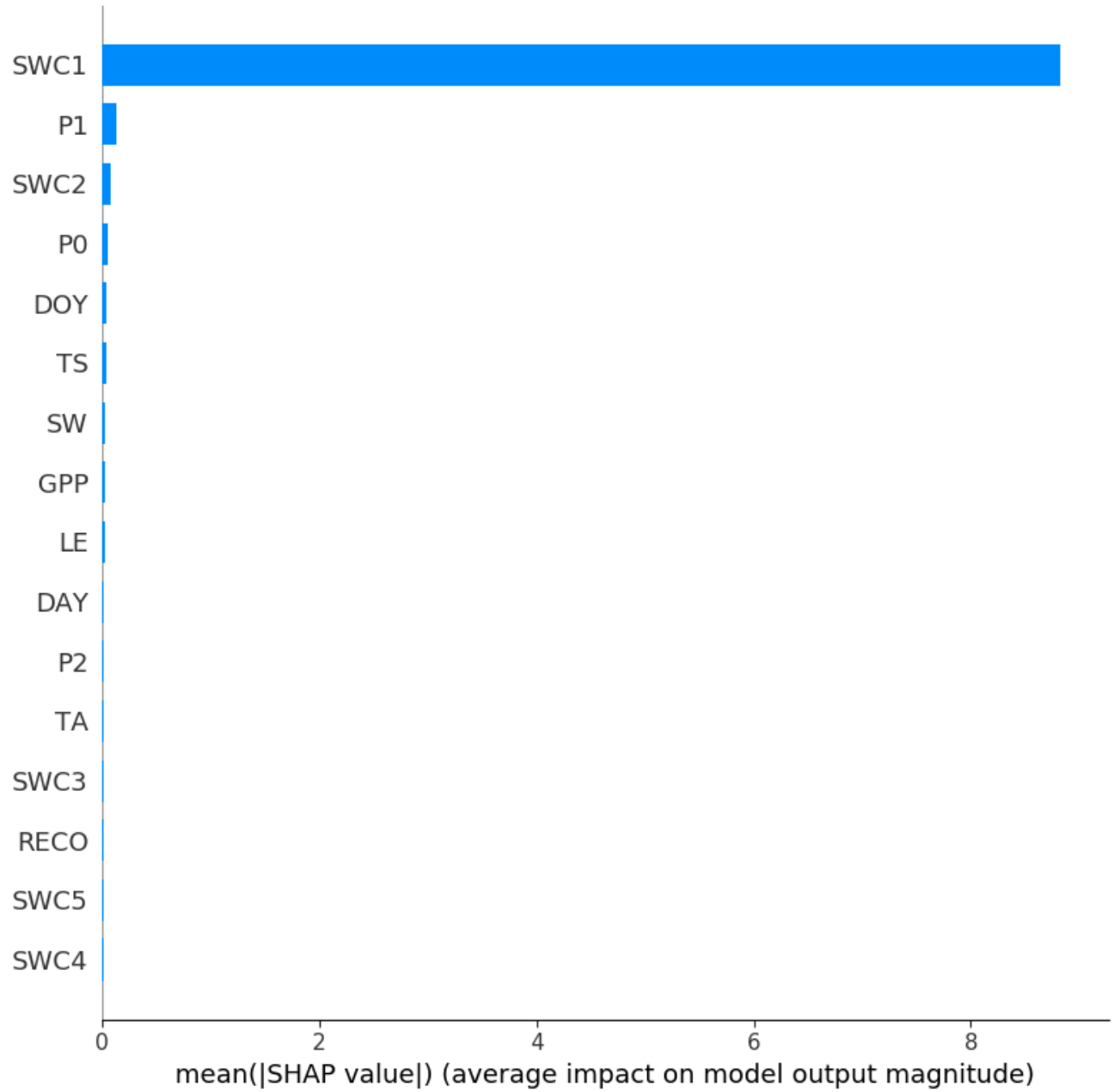
depend_shap() provides Shapley values distribution with TS variation.

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



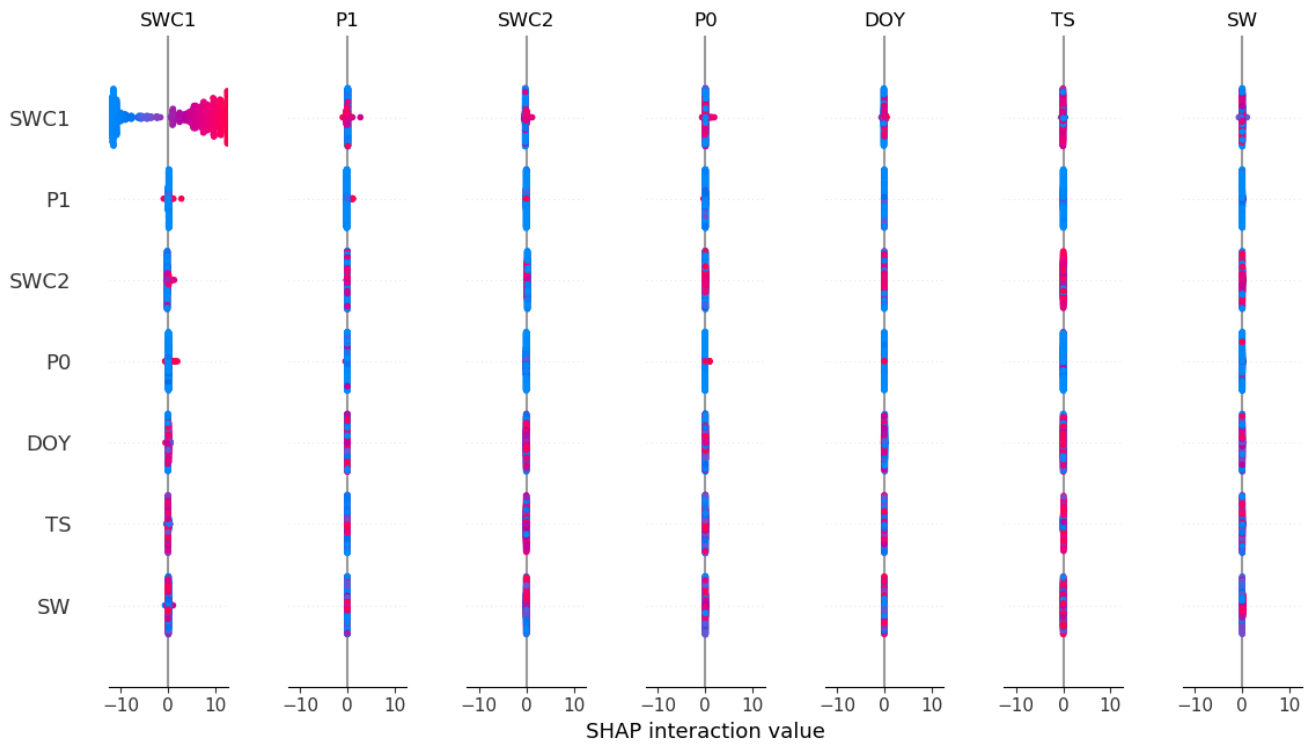
mean_shap() provides averaged Shapley values of features.

```
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.

###

Contributing

ExplainAI uses MIT license; contributions are welcome!

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

ExplainAI supports Python 2.7 and Python 3.4+ .

Changelog