1.import numpy as np

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

2.import pandas as pd

from sklearn.preprocessing import MinMaxScaler

data = pd.read\_excel(r'c: .xlsx')

na\_col = data.isnull().sum()[data.isnull().sum() > 0].index

for i in na\_col:

data[i] = data[i].fillna(data[i].mean())

x = data.drop("Input", axis=1)

y = data["Input"]

scaler1 = MinMaxScaler()

scaler2 = MinMaxScaler()

x\_scaled = scaler1.fit\_transform(x)

y\_scaled = scaler2.fit\_transform(y.values.reshape(-1, 1))

x\_scaled\_df = pd.DataFrame(x\_scaled, columns=x.columns)

y\_scaled\_df = pd.DataFrame(y\_scaled, columns=["Input"])

print("x\_scaled\_df shape:", x\_scaled\_df.shape)

print("y\_scaled\_df shape:", y\_scaled\_df.shape)

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(x\_scaled\_df,y\_scaled\_df,test\_size=0.2,random\_state=922)

from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor,AdaBoostRegressor

from hyperopt import hp, tpe, fmin,STATUS\_OK, Trials

from hyperopt.early\_stop import no\_progress\_loss

from sklearn.model\_selection import KFold,cross\_val\_score

from sklearn.tree import DecisionTreeRegressor

# RF CODE

def hyperopt\_objective(params):

    model=RandomForestRegressor(n\_estimators=int(params['n\_estimators'])

              ,max\_depth=int(params['max\_depth'])

              ,min\_samples\_leaf=int(params['min\_samples\_leaf'])

              ,min\_samples\_split=params['min\_samples\_split']

              ,random\_state=7

              )

    cv=KFold(n\_splits=5,shuffle=True,random\_state=7)

    validate\_loss=cross\_val\_score(model,xtrain,ytrain

                                   ,cv=cv

                                   ,scoring='r2'

                                   ,n\_jobs=-1

                                   ,error\_score='raise').mean()

    return -validate\_loss

n\_estimators\_range = range(10,1000,10)

max\_depth\_range = range(1,10,1)

min\_samples\_leaf\_range = range(1,10,1)

min\_samples\_split\_range = range(2,10,1)

param\_grid\_simple={'n\_estimators':hp.choice('n\_estimators',n\_estimators\_range)

                   ,'max\_depth':hp.choice('max\_depth',max\_depth\_range)

                   ,'min\_samples\_leaf':hp.choice('min\_samples\_leaf',min\_samples\_leaf\_range)

                   ,'min\_samples\_split':hp.choice('min\_samples\_split',min\_samples\_split\_range)

                  }

def param\_hyperopt(max\_evals=100):

    trials=Trials()

    early\_stop\_fn=no\_progress\_loss(100)

    params\_best=fmin(hyperopt\_objective

                     ,space=param\_grid\_simple

                     ,algo=tpe.suggest

                     ,max\_evals=max\_evals #

                     ,trials=trials

                     ,early\_stop\_fn=early\_stop\_fn

                    )

    print('best parmas:',params\_best)

    return params\_best,trials

print

params\_best,trials = param\_hyperopt(max\_evals=30)

print

best\_rf\_model = RandomForestRegressor(max\_depth = max\_depth\_range[params\_best['max\_depth']],

                                      n\_estimators = n\_estimators\_range[params\_best['n\_estimators']],

                                      min\_samples\_leaf = min\_samples\_leaf\_range[params\_best['min\_samples\_leaf']],

                                      min\_samples\_split = min\_samples\_split\_range[params\_best['min\_samples\_split']])

import joblib

joblib.dump(best\_rf\_model,'best\_rf.h5')

import joblib

best\_rf\_model = joblib.load('best\_rf.h5')

best\_rf\_model.fit(xtrain, ytrain)

ypred = best\_rf\_model.predict(xtest)

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

mse\_test = mean\_squared\_error(ytest,ypred)

mae\_test = mean\_absolute\_error(ytest,ypred)

rmse\_test = mean\_squared\_error(ytest,ypred,squared=False)

r2 = r2\_score(ytest,ypred)

print( mse\_test)

print( mae\_test)

print( rmse\_test)

print("R^2:", r2)

feature\_importances = best\_rf\_model.feature\_importances\_

sorted\_indices = np.argsort(feature\_importances)[::-1]

sorted\_importances = feature\_importances[sorted\_indices]

feature\_names = data.columns[1:]

sorted\_feature\_names = feature\_names[sorted\_indices]

for i in range(len(sorted\_indices)):

    print("{} ({})".format(sorted\_feature\_names[i], sorted\_importances[i]))

import pandas as pd

feature\_importance\_df = pd.DataFrame({'Feature': sorted\_feature\_names, 'Importance': sorted\_importances})

feature\_importance\_df.to\_excel(r'c:.xlsx', index=False)

base\_features = x.columns.values.tolist()

base\_features

import joblib

best\_rf\_model = joblib.load('best\_rf.h5')

max\_depth = best\_rf\_model.get\_params()['max\_depth']

print( max\_depth)

import joblib

best\_rf\_model = joblib.load('best\_rf.h5')

n\_estimators = best\_rf\_model.get\_params()['n\_estimators']

print( n\_estimators)

# Adaboost CODE

##Adaboost parameter optimization

def hyperopt\_objective(params):

    model=AdaBoostRegressor(n\_estimators=int(params['n\_estimators'])

              ,learning\_rate=params['learning\_rate']

              )

    cv=KFold(n\_splits=5,shuffle=True,random\_state=7)

    validate\_loss=cross\_val\_score(model,xtrain,ytrain

                                   ,cv=cv

                                   ,scoring='r2'

                                   ,n\_jobs=-1

                                   ,error\_score='raise').mean()

    return -validate\_loss

n\_estimators\_range = range(10,100,10)

param\_grid\_simple={'n\_estimators':hp.choice('n\_estimators',n\_estimators\_range)

                   ,'learning\_rate':hp.quniform('learning\_rate',0.01,1,0.01)

                  }

def param\_hyperopt(max\_evals=100):

    trials=Trials()

    early\_stop\_fn=no\_progress\_loss(100)

    params\_best=fmin(hyperopt\_objective

                     ,space=param\_grid\_simple

                     ,algo=tpe.suggest

                     ,max\_evals=max\_evals

                     ,trials=trials

                     ,early\_stop\_fn=early\_stop\_fn

                    )

    print('best parmas:',params\_best)

    return params\_best,trials

print()

params\_best,trials = param\_hyperopt(max\_evals=50)

print()

best\_adat\_model = AdaBoostRegressor(n\_estimators = n\_estimators\_range[params\_best['n\_estimators']],

                                    learning\_rate = params\_best['learning\_rate'])

import joblib

joblib.dump(best\_adat\_model,'best\_adat.h5')

best\_adat\_model = joblib.load('best\_adat.h5')

import joblib

best\_adat\_model = joblib.load('best\_adat.h5')

n\_estimators = best\_adat\_model.get\_params()['n\_estimators']

print(n\_estimators)

import joblib

best\_adat\_model = joblib.load('best\_adat.h5')

learning\_rate = best\_adat\_model.get\_params()['learning\_rate']

print(learning\_rate)

best\_adat\_model.fit(xtrain, ytrain)

ypred = best\_adat\_model.predict(xtest)

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

mse\_test = mean\_squared\_error(ytest,ypred)

mae\_test = mean\_absolute\_error(ytest,ypred)

rmse\_test = mean\_squared\_error(ytest,ypred,squared=False)

r2 = r2\_score(ytest,ypred)

print( mse\_test)

print( mae\_test)

print( rmse\_test)

print("R^2:", r2)

# GBDT CODE

def hyperopt\_objective(params):

    model=GradientBoostingRegressor(n\_estimators=int(params['n\_estimators'])

              ,max\_depth=int(params['max\_depth'])

              ,min\_samples\_leaf=int(params['min\_samples\_leaf'])

              ,min\_samples\_split=params['min\_samples\_split']

              ,random\_state=7

              )

    cv=KFold(n\_splits=5,shuffle=True,random\_state=7)

    validate\_loss=cross\_val\_score(model,xtrain,ytrain

                                   ,cv=cv

                                   ,scoring='r2'

                                   ,n\_jobs=-1

                                   ,error\_score='raise').mean()

    return -validate\_loss

n\_estimators\_range = range(10,1000,10)

max\_depth\_range = range(1,10,1)

min\_samples\_leaf\_range = range(1,10,1)

min\_samples\_split\_range = range(2,10,1)

param\_grid\_simple={'n\_estimators':hp.choice('n\_estimators',n\_estimators\_range)

                   ,'max\_depth':hp.choice('max\_depth',max\_depth\_range)

                   ,'min\_samples\_leaf':hp.choice('min\_samples\_leaf',min\_samples\_leaf\_range)

                   ,'min\_samples\_split':hp.choice('min\_samples\_split',min\_samples\_split\_range)

                  }

def param\_hyperopt(max\_evals=100):

    trials=Trials()

    early\_stop\_fn=no\_progress\_loss(100)

    params\_best=fmin(hyperopt\_objective

                     ,space=param\_grid\_simple

                     ,algo=tpe.suggest

                     ,max\_evals=max\_evals

                     ,trials=trials

                     ,early\_stop\_fn=early\_stop\_fn

                    )

    print('best parmas:',params\_best)

    return params\_best,trials

print()

params\_best,trials = param\_hyperopt(max\_evals=30)

print()

best\_gbdt\_model = GradientBoostingRegressor(max\_depth = max\_depth\_range[params\_best['max\_depth']],

                                      n\_estimators = n\_estimators\_range[params\_best['n\_estimators']],

                                      min\_samples\_leaf = min\_samples\_leaf\_range[params\_best['min\_samples\_leaf']],

                                      min\_samples\_split = min\_samples\_split\_range[params\_best['min\_samples\_split']])

import joblib

joblib.dump(best\_gbdt\_model,'best\_gbdt.h5')

best\_gbdt\_model = joblib.load('best\_gbdt.h5')

import joblib

best\_gbdt\_model = joblib.load('best\_gbdt.h5')

n\_estimators = best\_gbdt\_model.get\_params()['n\_estimators']

print( n\_estimators)

import joblib

best\_gbdt\_model = joblib.load('best\_gbdt.h5')

max\_depth = best\_gbdt\_model.get\_params()['max\_depth']

print( max\_depth)

import joblib

best\_gbdt\_model = joblib.load('best\_gbdt.h5')

learning\_rate = best\_gbdt\_model.get\_params()['learning\_rate']

print( learning\_rate)

import joblib

best\_gbdt\_model = joblib.load('best\_gbdt.h5')

min\_samples\_split = best\_gbdt\_model.get\_params()['min\_samples\_split']

print( min\_samples\_split)

best\_gbdt\_model.fit(xtrain, ytrain)

ypred = best\_gbdt\_model.predict(xtest)

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

mse\_test = mean\_squared\_error(ytest,ypred)

mae\_test = mean\_absolute\_error(ytest,ypred)

rmse\_test = mean\_squared\_error(ytest,ypred,squared=False)

r2 = r2\_score(ytest,ypred)

print( mse\_test)

print( mae\_test)

print( rmse\_test)

print("R^2:", r2)

# SVM CODE

from sklearn.svm import SVR

def hyperopt\_objective(params):

    model=SVR(C=params['C']

              ,degree=int(params['degree'])

              ,epsilon=params['epsilon']

              )

    cv=KFold(n\_splits=5,shuffle=True,random\_state=7)

    validate\_loss=cross\_val\_score(model,xtrain,ytrain

                                   ,cv=cv

                                   ,scoring='r2'

                                   ,n\_jobs=-1

                                   ,error\_score='raise').mean()

    return -validate\_loss

degree\_range = range(1,10,1)

param\_grid\_simple={'C':hp.quniform('C',0.1,10.0,0.1)

                   ,'degree':hp.choice('degree',degree\_range)

                   ,'epsilon':hp.quniform('epsilon',0.01,0.3,0.01)

                  }

def param\_hyperopt(max\_evals=100):

    trials=Trials()

    early\_stop\_fn=no\_progress\_loss(100)

    params\_best=fmin(hyperopt\_objective

                     ,space=param\_grid\_simple

                     ,algo=tpe.suggest

                     ,max\_evals=max\_evals

                     ,trials=trials

                     ,early\_stop\_fn=early\_stop\_fn

                    )

    print('best parmas:',params\_best)

    return params\_best,trials

print()

params\_best,trials = param\_hyperopt(max\_evals=100)

print()

best\_svr\_model = SVR(degree = degree\_range[params\_best['degree']],

                      C = 0.1,epsilon = 0.03)

import joblib

joblib.dump(best\_svr\_model,'best\_svr.h5')

import joblib

best\_svr\_model = joblib.load('best\_svr.h5')

best\_svr\_model.fit(xtrain, ytrain)

ypred = best\_svr\_model.predict(xtest)

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

mse\_test = mean\_squared\_error(ytest,ypred)

mae\_test = mean\_absolute\_error(ytest,ypred)

rmse\_test = mean\_squared\_error(ytest,ypred,squared=False)

r2 = r2\_score(ytest,ypred)

print( mse\_test)

print( mae\_test)

print( rmse\_test)

print("R^2:", r2)

# SHAP CODE

import shap

import matplotlib.pyplot as plt

from sklearn.inspection import PartialDependenceDisplay

from time import time

shap.initjs

explainer = shap.TreeExplainer(best\_rf\_model)

shap\_values = explainer.shap\_values(xtest)

plt.figure(dpi=400)

# fig = plt.gcf()

shap.summary\_plot(shap\_values, xtest, show=False)

plt.savefig(r'c: .png', dpi=1200)

import shap

import numpy as np

import matplotlib.pyplot as plt

from sklearn.inspection import PartialDependenceDisplay

from time import time

shap.initjs()

explainer = shap.TreeExplainer(best\_rf\_model)

shap\_values = explainer.shap\_values(xtest)

plt.figure(dpi=400)

shap\_values\_rounded = np.round(shap\_values, 2)

shap.summary\_plot(shap\_values\_rounded, xtest, show=False)

shap\_interaction\_values = explainer.shap\_interaction\_values(xtest)

shap.summary\_plot(shap\_interaction\_values, xtest,feature\_names=feature\_names, show=False)

feature\_names = data.columns[1:]

plt.savefig(r'c: .png', dpi=1200)

shap.partial\_dependence\_plot("FA", best\_rf\_model.predict, xtest, ice=False, model\_expected\_value=True, feature\_expected\_value=True, show=False)

plt.savefig(r'c:.png', dpi=1200)

shap.partial\_dependence\_plot("FA", best\_rf\_model.predict, xtest, ice=True, model\_expected\_value=True, feature\_expected\_value=True)

plt.savefig(r'c:.png', dpi=1200)

from time import time

from sklearn.inspection import PartialDependenceDisplay

common\_params = {

    "subsample": 50,

    "n\_jobs": 2,

    "grid\_resolution": 20,

    "random\_state": 0,

}

tic = time()

\_, ax = plt.subplots(ncols=2, figsize=(6, 4), sharey=True, constrained\_layout=True)

features\_info = {

    "features": ["FA", "GDP"],

    "kind": "both",

    "centered": True,

}

display = PartialDependenceDisplay.from\_estimator(

    best\_rf\_model,

    xtrain,

    \*\*features\_info,

    ax=ax,

    \*\*common\_params,

)

print(f"done in {time() - tic:.3f}s")

\_ = display.figure\_.suptitle("ICE and PDP representations", fontsize=16)

plt.savefig(r'c:.png', dpi=1200)

features\_info = {

    "features": ["CLA", "LP", ("CLA", "LP")],

    "kind": "average",

}

\_, ax = plt.subplots(ncols=3, figsize=(10, 4), constrained\_layout=True)

tic = time()

display = PartialDependenceDisplay.from\_estimator(

    best\_rf\_model,

    xtrain,

    \*\*features\_info,

    ax=ax,

    \*\*common\_params,

)

print(f"done in {time() - tic:.3f}s")

\_ = display.figure\_.suptitle(

    "1-way vs 2-way of numerical PDP using gradient boosting", fontsize=16

)

plt.savefig(r'c:.png', dpi=1200)

feature\_of\_interest = 'FA'

interaction\_feature = 'GDP'

shap.dependence\_plot(feature\_of\_interest, shap\_values, xtest,

                     feature\_names=feature\_names, interaction\_index=interaction\_feature,

                     show=False)

plt.savefig(r'c:.png', dpi=1200)

import shap

shap.plots.force(explainer.expected\_value, shap\_values[5,:], xtest.iloc[5, :], show=False, matplotlib = True)

plt.savefig(r'c:.png', bbox\_inches='tight', dpi=1200)

import shap

import matplotlib.pyplot as plt

import numpy as np

formatted\_xtest = np.round(xtest, 2)

fig = shap.plots.force(explainer.expected\_value, shap\_values[5,:], formatted\_xtest.iloc[5, :], show=False, matplotlib=True)

ax = fig.axes[0]

ax.set\_xticklabels(["{:.2g}".format(label) for label in ax.get\_xticks()])

ax.set\_yticklabels(["{:.2g}".format(label) for label in ax.get\_yticks()])

plt.savefig(r'c:.png', bbox\_inches='tight', dpi=1200)

plt.show()

import shap

import numpy as np

import matplotlib.pyplot as plt

from sklearn.inspection import PartialDependenceDisplay

from time import time

shap.initjs()

data\_for\_prediction = xtest.iloc[5, :]

data\_for\_prediction = np.around(data\_for\_prediction, decimals=2)

if len(explainer.expected\_value.shape) > 0:

    expected\_value = explainer.expected\_value[0]

    expected\_value = explainer.expected\_value

shap\_values\_2d = np.array([shap\_values[0]])  # 使用 [0] 索引创建二维数组

feature\_names = data.columns[1:]

shap.force\_plot(expected\_value, shap\_values\_2d, data\_for\_prediction, feature\_names=feature\_names, show=False, matplotlib=True)

plt.savefig(r'd:.png', bbox\_inches='tight', dpi=1200)

explainer = shap.Explainer(best\_rf\_model)

shap\_values = explainer(xtest)

shap.plots.heatmap(shap\_values, show=False)

plt.savefig(r'c:.png', dpi=1200)