支持向量机（SVM）：

支持向量机（Support Vector Machine, SVM）是一种强大的机器学习方法，用于分类和回归任务。在回归任务中，支持向量机的变体被称为支持向量回归（Support Vector Regression, SVR）。

代码示例：

```
import numpy as np
import pandas as pd
import time
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

plt.rcParams['font.sans-serif'] = ['SimHei', 'Songti SC', 'STFangsong']
plt.rcParams['axes.unicode_minus'] = False

xx =
['wob_min','wob_max','wob_mean','wob_std','wob_skew','wob_kuet','T_min','T_max'
,'T_mean','T_std','T_skew','T_kuet']

# 加载数据集
X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 数据预处理：对特征进行标准化
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 创建SVR模型
# 注意：SVR的默认核函数是RBF（径向基函数），你可以通过kernel参数更改核函数
svr_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)

# 训练模型
svr_model.fit(X_train_scaled, y_train)

# 进行预测
y_pred = svr_model.predict(X_test_scaled)

# 评估模型
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
r2score = r2_score(y_test, y_pred)
bst_rmse = mean_squared_error(y_test, y_pred) ** 0.5
```

```
print("均方误差(MSE):", mse)
print('R方: ', r2score)
print('rmse:', bst_rmse)
```

```
均方误差(MSE): 783.1521620608822
R方:  0.6557053709771413
rmse: 27.984855941399488
```

随机森林(random forest)：

随机森林进行回归分析是机器学习中的一个常见任务，可以有效地处理各种类型的回归问题。

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error

# 加载数据集
X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建随机森林回归模型
# n_estimators表示树的数量，max_depth表示树的最大深度
rf_regressor = RandomForestRegressor(n_estimators=100, max_depth=None,
random_state=42)

# 训练模型
rf_regressor.fit(X_train, y_train)

# 使用模型进行预测
y_pred = rf_regressor.predict(X_test)

# 评估模型
mse = mean_squared_error(y_test, y_pred)
r2score = r2_score(y_test, y_pred)
bst_rmse = mean_squared_error(y_test, y_pred) ** 0.5
print("均方误差(MSE):", mse)
print('R方: ', r2score)
print('rmse:', bst_rmse)
```

```
均方误差(MSE): 43.34981460072534
R方:  0.9809422624884407
rmse: 6.5840576091590615
```

GBDT :

梯度提升决策树（Gradient Boosted Decision Trees，GBDT）是一种强大的回归和分类模型，它通过迭代地训练决策树来最小化损失函数。

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error

# 加载数据集
X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建GBDT回归模型
# n_estimators表示要训练的树的数量，max_depth表示每棵树的最大深度
gbdt_regressor = GradientBoostingRegressor(n_estimators=100, max_depth=3,
random_state=42)

# 训练模型
gbdt_regressor.fit(X_train, y_train)

# 使用模型进行预测
y_pred = gbdt_regressor.predict(X_test)

# 评估模型
mse = mean_squared_error(y_test, y_pred)
r2score = r2_score(y_test, y_pred)
bst_rmse = mean_squared_error(y_test, y_pred) ** 0.5
print("均方误差(MSE):", mse)
print('R方: ', r2score)
print('rmse:', bst_rmse)
```

```
均方误差(MSE): 66.75472824964271
R方:  0.9706528366878904
rmse: 8.170356678238882
```

Adboost:

AdaBoost（Adaptive Boosting）是一种集成学习方法，通过将多个弱学习器组合成一个强学习器来提高模型的预测性能。

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor

# 加载数据集
X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建AdaBoost回归模型，使用决策树作为基学习器
# n_estimators表示要训练的弱学习器的数量
ada_regressor =
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=4),
                                  n_estimators=100, random_state=42)

# 训练模型
ada_regressor.fit(X_train, y_train)

# 使用模型进行预测
y_pred = ada_regressor.predict(X_test)

# 评估模型
mse = mean_squared_error(y_test, y_pred)
r2score = r2_score(y_test, y_pred)
bst_rmse = mean_squared_error(y_test, y_pred) ** 0.5
print("均方误差(MSE):", mse)
print('R方: ', r2score)
print('rmse:', bst_rmse)
```

```
均方误差(MSE): 67.4506798966619
R方:  0.9703468777366978
rmse: 8.21283628819313
```

LightGBM：

LightGBM是一个高效的梯度提升框架，由微软提供支持，它设计用于分布式和高效计算，尤其适合处理大规模数据。与其他梯度提升库相比，LightGBM具有更快的训练速度和更高的效率，同时还支持分类、回归及排名任务。

```
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error

# 加载数据集
X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]

# 划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建LightGBM数据格式
train_data = lgb.Dataset(X_train, label=y_train)
test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

# 设置模型参数
params = {
    'boosting_type': 'gbdt',  # 传统的梯度提升决策树
    'objective': 'regression',  # 回归任务
    'metric': 'l2',  # 评价指标，l2表示使用L2损失（均方误差MSE）
    'num_leaves': 31,  # 叶子节点数
    'learning_rate': 0.05,  # 学习率
    'feature_fraction': 0.9,  # 建树的特征选择比例
    'bagging_fraction': 0.8,  # 建树的样本采样比例
    'bagging_freq': 5,  # k表示每k次迭代执行bagging
    'verbose': 0  # <0 显示致命的，=0 显示错误（警告），>0 显示信息
}

# 训练模型
gbm = lgb.train(params, train_data, num_boost_round=100, valid_sets=
[train_data, test_data], early_stopping_rounds=10)

# 使用最佳迭代次数进行预测
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)

# 评估模型
mse = mean_squared_error(y_test, y_pred)
r2score = r2_score(y_test, y_pred)
bst_rmse = mean_squared_error(y_test, y_pred) ** 0.5
print("均方误差(MSE):", mse)
print('R方: ', r2score)
print('rmse:', bst_rmse)
```

```
均方误差(MSE): 37.86915708497443
R方:  0.9833517060647966
rmse: 6.1537920898397624
```

XGBoost：

```
# xgboost回归器
import xgboost as xgb
from xgboost import plot_importance,plot_tree
import numpy as np
import pandas as pd
import time
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
plt.rcParams['font.sans-serif'] = ['SimHei', 'Songti SC', 'STFangsong']
plt.rcParams['axes.unicode_minus'] = False


xx =
['wob_min','wob_max','wob_mean','wob_std','wob_skew','wob_kuet','T_min','T_max'
,'T_mean','T_std','T_skew','T_kuet']

X = res[xx]
## Y为要预测的数值
y = res["静态抗压强度σc/MPa"]
feature_name =X.columns
#将数据分割训练数据与测试数据
from sklearn.model_selection import train_test_split
# 随机采样20%的数据构建测试样本，其余作为训练样本
Xtrain, Xtest, ytrain, ytest = train_test_split(X,
y,test_size=0.2,random_state=1)

dtrain = xgb.DMatrix(Xtrain, ytrain,feature_names=feature_name)
dtest = xgb.DMatrix(Xtest,feature_names=feature_name)

params = {'booster':'gbtree',
          'objective':'reg:squarederror',
          'eta':0.1,
          'gamma':0,
          'alpha':0,
          'lambda':3,
          'max_depth':6,
          'subsample':0.7,
          'colsample_bytree':0.9,
          'min_child_weight':1,
          'learning_rate':0.1,
          'seed':1000,
          'nthread':1
          }
num_round = 500

start_time = time.time()
bst = xgb.train(params, dtrain, num_round)
```

```
end_time = time.time()
print('The training time = {}'.format(end_time - start_time))

bst_ypred = bst.predict(dtest)
r2score = r2_score(ytest, bst_ypred)
bst_rmse = mean_squared_error(ytest, bst_ypred) ** 0.5

print('The R2_score = {}'.format(r2score))
print('The rmse of prediction is:', bst_rmse)
#plot_importance(bst,importance_type="weight")
plt.figure()
plt.plot(range(60), bst_ypred[:60], c="g", label="ypred", linewidth=2)
plt.plot(range(60), ytest[:60], c="r", label="ytest", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("静态抗压强度σc/MPa")
plt.show()
plt.show()
```

```
The training time = 1.079322099685669
The R2_score = 0.9881620836493739
The rmse of prediction is: 5.277385320014107
```

使用遗传算法优化的xgboost模型：

遗传算法优化

遗传算法是一种基于自然选择和遗传学原理的搜索启发式算法，可以用来优化决策树的结构和参数。本研究将遗传算法用于：

- 选择最佳的特征和特征分裂点。
- 优化决策树的深度和复杂性，防止过拟合。
- 选择最佳的集成模型参数，如随机森林中树的数量、GBDT中的学习率等。

通过遗传算法，可以定义一个适应度函数（比如，交叉验证下的模型准确率），然后不断迭代，通过选择（选择最佳模型）、交叉（组合最佳模型的参数）和变异（随机改变参数），来寻找最优解。

```
from sklearn.metrics import mean_squared_error
from deap import base, creator, tools, algorithms
import random

def evalXGBoost(params):
    max_depth, eta, subsample, colsample_bytree = params
    params = {
        'objective':'reg:squarederror',
        'max_depth': int(max_depth),
        'eta': eta,
        'subsample': subsample,
        'colsample_bytree': colsample_bytree,
        'learning_rate': 0.1,
        'seed': 1000,
        'nthread': 1,

        'booster':'gbtree',
          'gamma':0,
          'alpha':0,
          'lambda':3,
          'min_child_weight':1,

    }
    dtrain = xgb.DMatrix(Xtrain, ytrain)

    num_boost_round = 50
    nfold = 5
    metrics = {'rmse'}
    seed = 0
    verbose_eval = True  # 控制是否打印每次迭代的评估结果
    # 执行交叉验证
    cv_result = xgb.cv(params=params,
                       dtrain=dtrain,
                       num_boost_round=num_boost_round,
                       nfold=nfold,
                       metrics=metrics,
                       seed=seed,
                       verbose_eval=verbose_eval)  # 设置 verbose_eval=True 即
可输出评估结果

    # 返回RMSE的负值
    return (-cv_result['test-rmse-mean'].iloc[-1],)  # 确保以元组形式返回




creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

```
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_max_depth", random.randint, 3, 10)
toolbox.register("attr_eta", random.uniform, 0.01, 0.3)
toolbox.register("attr_subsample", random.uniform, 0.5, 1)
toolbox.register("attr_colsample_bytree", random.uniform, 0.5, 1)
toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_max_depth, toolbox.attr_eta,
toolbox.attr_subsample, toolbox.attr_colsample_bytree), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", evalXGBoost)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
#toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
# 自定义变异函数，确保colsample_bytree在[0,1]范围内
def custom_mutate(individual, indpb):
    if random.random() < indpb:
        # 假设individual中的某个位置对应于colsample_bytree参数
        individual[3] = random.uniform(0, 1)  # 以3为例，表示colsample_bytree参数
的位置
    return individual,

# 使用自定义的变异函数
toolbox.register("mutate", custom_mutate, indpb=0.2)
def custom_crossover(ind1, ind2):
    # 交叉操作的示例逻辑
    # 确保在交叉后，colsample_bytree的值处于[0,1]范围内
    ind1[3] = min(1, max(0, ind1[3]))
    ind2[3] = min(1, max(0, ind2[3]))
    return ind1, ind2

toolbox.register("mate", custom_crossover)


population = toolbox.population(n=50)


result = algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
                             stats=None, halloffame=None, verbose=True)
```

```
最优参数:
[3, 0.13486291954630472, 0.6227066482028191, 0.11907194689374956]
最优参数的性能（负RMSE）：(-15.68715096915064,)
运行时间: 320.3152508735657秒
```

cnn+bp

代码：

```
#cnn加bp
import numpy as np
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

rock_character = pd.read_excel('../data/模型数据副本.xlsx',sheet_name = '材料',
engine='openpyxl')
device_wob = pd.read_excel('../data/模型数据副本.xlsx',sheet_name = '设备wob',
engine='openpyxl')
device_T = pd.read_excel('../data/模型数据副本.xlsx',sheet_name = '设备T',
engine='openpyxl')
device_wob['time'] = [int(temp) for temp in device_wob['时间/s']]
device_T['time'] = [int(temp) for temp in device_T['时间/s']]
device_S = pd.read_excel('../data/模型数据副本.xlsx',sheet_name = '设备S',
engine='openpyxl')
device_Z = pd.read_excel('../data/模型数据副本.xlsx',sheet_name = '设备Z',
engine='openpyxl')
del device_wob['时间/s']
del device_T['时间/s']
columns = ['材料2','材料4','材料5','材料7','材料8','材料10','材料11','材料13','材料
55','材料21','材料22']
res_df = pd.DataFrame(columns=
['time1','time2','time3','time4','time5','time6','time7','time8','time9','time1
0','res'])
T_samples = []
wob_samples = []
outputs = []
for time in range(0,len(device_T)-1,10):
    device_wob_temp = device_wob[time:time+10]
    device_T_temp = device_T[time:time+10]
    for column in columns:
        T_samples.append(device_T_temp[column].values)
        wob_samples.append(device_wob_temp[column].values)
        output = rock_character[rock_character['岩性名称']==column]['静态抗压强
度'].values[0]
        outputs.append(output)

# 将T和wob样本转换为多通道输入
T_data = np.array(T_samples).reshape(-1, len(T_samples[0]), 1)  # (样本数, 时间步
长, 1)
```

```
wob_data = np.array(wob_samples).reshape(-1, len(wob_samples[0]), 1)  # (样本数, 时间步长, 1)

# 将T和wob拼接为一个多通道输入
input_data = np.concatenate((T_data, wob_data), axis=-1)  # 形状变为(样本数, 时间步长, 特征数*2)

# 目标输出整理为二维数组
output_data = np.array(outputs).reshape(-1, 1)



# 构建模型
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=input_data.shape[1:]))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='linear'))  # 对于回归问题，最后的输出层通常使用线性激活函数

model.compile(loss='mean_squared_error', optimizer='adam')  # 使用均方误差损失函数和Adam优化器

#划分训练集和测试集

X_train, X_test, y_train, y_test = train_test_split(input_data, output_data, test_size=0.2, random_state=42)
# 训练模型
model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=0)

# 预测
prediction = model.predict(X_test)
#评估模型精度mse、rmse、r2

mse = mean_squared_error(y_test, prediction)
print("均方误差: ", mse)
rmse = np.sqrt(mse)
print("均方根误差: ", rmse)

r2 = r2_score(y_test, prediction)
print("R2: ", r2)
```

```
均方误差:  142.47288206825402
均方根误差:  11.936200487100324
R2:  0.9361194793222453
```

相关性分析:

去除相关性大于80%的变量

设备S:

- 设备Z:静态抗压强度、回弹均值、滑动摩擦系数、声级、孔隙度、标定温度

静态抗压强度:设备Z、回弹均值、动态强度、滑动摩擦系数、声级

弹性模量:回弹均值

泊松比:

抗拉强度:

粘聚力:

内摩擦角:

- 回弹均值
- 动态强度
- 滑动摩擦系数
- 声级

波速:

密度均值:

- 渗透率:孔隙度

孔隙度:设备Z、回弹均值、渗透率

标定温度:设备Z

- 联合分布图右上角颜色;

- 整理图表的数据；
- 整理一个文档，说明使用的算法；