```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  6 10:42:31 2018

Demo version to show the usage of Random Forest ML

* Get the data
* Clean and prepare the data
* Prepare the features and samples
* Select the training and testing sets
* Train the model
* Test the model

@author: hakalam7
"""


# Load necessary packages
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import platform



def cross_val(model, X, target, featureNames, iprint=True):
    from sklearn.cross_validation import KFold #For K-fold cross validation
    """
    General help function to do cross-validation
    model  = classifier which has the method predict: model.predict(X)
    X      = input samples, for example X = features_test
    target = known target values
    """

    # n-fold cross-validation
    n_folds=5
    kf = KFold(target.shape[0], n_folds=n_folds, shuffle=True)
    error = []
    for train, test in kf:
        train_predictors = (X[train,:])
        train_target = target[train]
        if(False):
            print('predictors:', train_predictors, 'shape:', train_predictors.shape, '\n')
            print('target:', train_target)
        model.fit(train_predictors, train_target)
        error.append(model.score(X[test,:], target[test]))
    score = np.round(  np.mean(error), 4  )
    score_error_of_mean = np.round(  np.std(error, ddof=1) / np.sqrt(len(error)), 4  )
    std = np.round(np.std(error, ddof=1), 4)

    if(iprint):
        print('*Score (cross-validation):', score, '+-', score_error_of_mean, '(std: ', std, ')', 'folds:', n_folds, ',
total data:', len(target), ', validat. data:', len(test))

    if(False):
        print('error in the folds:', np.round(error, 2))

    return(score, score_error_of_mean)



def main():

    # Get the data
    print('---')
    rawdata="../../../Documents/Datasets/masterdata.dat"
    dfraw = pd.read_csv(rawdata)

    print("Data types in dataframe:\n",dfraw.dtypes)
    if(True):
        print("Head of original:\n",dfraw.head(6))



    # Randomize the ordering of rows and print a sample
    print('---')
    ishuffle=True
    if (ishuffle):
        np.random.seed(163519) # rand
        #
        print('Keep the intial shuffle fixed (always the same, with seed(X))')
        print('- important since we want a frozen, always same test set')
        df=dfraw.reindex(np.random.permutation(dfraw.index))
        df=df.reset_index(drop=True)
        if(True):
            print("Head of shuffled:\n",df.head(4))
        print("*Data frame rows shufffeatureNames=['Type','Nval','Nn','Coord']led")
        print("df[0]:",df['Hads'][0])
        np.random.seed(None)
    print('Finished reading data, length of data:',len(df))


    # Describe the data
    print('---')
    print(df.describe())


    # Drop data points that we don't use for modelling
    print('---')
    print('Drop cases |Hads| > 6.0')
    df = df[df.Hads < 6.0]

    print('Drop cases Z = 0 (vacancies)')
    df = df[df.Z > 0]

    print('Drop cases q != 0')
    df = df[df.q == 0]


    # Add a new feature, nominal 'Coord' in the dataframe
    # - for cases 0,2,3,6 it is 6
    # - for case 5 it is 5
    # - for case 1,4 it is 4
    print('---')
    df.loc[ df['Type'] == 0 ,'Coord'] = 6
    df.loc[ df['Type'] == 2 ,'Coord'] = 6
    df.loc[ df['Type'] == 3 ,'Coord'] = 6
    df.loc[ df['Type'] == 6 ,'Coord'] = 6
    df.loc[ df['Type'] == 5 ,'Coord'] = 5
    df.loc[ df['Type'] == 1 ,'Coord'] = 4
    df.loc[ df['Type'] == 4 ,'Coord'] = 4
```

```
# Select the features
featureNames=['Type','Nval','Nn','Coord']
print(' \nFeature names:',featureNames)
nFeatures=len(featureNames)
nSamples=len(df)
features=np.zeros((nSamples,nFeatures))
i=-1
for featureName in featureNames:
    i+=1
    features[:,i]=df[featureName].values


# Set target numerical: yNum
print('---')
print('Set target values')
yNum=df['Hads'].values



# Set target binary: yBin (a new variable)
HadsLim=0.5
# Create a new column 'HadsBin' in the dataframe
df.loc[ np.abs(df['Hads']) > HadsLim  ,'HadsBin'] = 0
df.loc[ np.abs(df['Hads']) <= HadsLim  ,'HadsBin'] = 1

yBin=df['HadsBin'].values
yBin=yBin.astype(int)


# Auxiliary function to return training and test sets
def selectsets(data, sizeTestSet):
    trainingSet=data[:-sizeTestSet]
    testSet=data[-sizeTestSet:]
    return(trainingSet,testSet)


# Set training and test sets
print('---')
sizeTestSet = 14
print("\n*Size of the test set:",sizeTestSet)
```

```
features_train, features_test = selectsets(features, sizeTestSet)
y_train, y_test = selectsets(yBin, sizeTestSet)
y_train_num, y_test_num = selectsets(yNum, sizeTestSet)
print("Test set binary:", y_test)
print("Test set numeric:", y_test_num)

if(True):
    print("Test set features, energy:")
    for i in range(sizeTestSet):
        print(features_test[i, :], y_test_num[i])



# Choose the machine learning model
print('---')
print('Predicting the class label 0/1 (are we in the studid range +-', HadsLim,'eV ?')
print('Choosing the machine learning model')
randomForest = False
if (randomForest):
    print('Random Forest:')
    clf = RandomForestClassifier(n_estimators=200, max_features="auto", oob_score=True, verbose=0, random_state=None)
else:
    print('Logistic regression:')
    clf = LogisticRegression()


# Train the model
clf.fit(features_train, y_train)


# In real case one needs to optimize the hyperparameters of the machine learning model
print('---')
print('Note: One should optimize the parameters of the Random Forest model - Skipped in this example')


# How well do we do on the training set?
print('---')
print('*Score (training set):',clf.score(features_train, y_train))

# How well do we do on the test set?
print('*Score (test set):',clf.score(features_test, y_test))
```

```
    # Cross validation
    doCrossValidationData = False
    if(doCrossValidationData):
        print("CROSS-VALIDATION DATA:")
        scores=[]
        CVrounds = 5
        for i in range(CVrounds):
            score = cross_val(clf, features_train, y_train, featureNames, iprint=False)
            scores.append(score[0])
            print('cross val round, score:', i, score)
        std = np.round(np.std(scores, ddof=1), 4)
        sem = std / np.sqrt(len(scores))  # standard error of the mean
        print('Scores:', scores)
        print('*Ave, std, std-of-mean of CV scores:', np.mean(scores), '/', std, '/', sem)
    else:
        print('*Cross-validation data part is skipped (doCrossValidationData = False) ')



    # Predicting numerical values - Regression model with Random Forest
    print('---')
    print('Predicting numerical values')
    reg = RandomForestRegressor(n_estimators=200, oob_score=True)
    reg.fit(features_train,y_train_num)
    print('*Score (training set) R^2:',reg.score(features_train, y_train_num))
    print('*Score (test set) R^2:',reg.score(features_test, y_test_num))




if __name__ == '__main__':
    main()
```