```
#Theodore McCullough, Copyright 2022
#Licensed Under the MIT License<https://opensource.org/licenses/MIT>

import sklearn
import numpy as np
import csv
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
import joblib
import matplotlib.pyplot as plt
import itertools
from itertools import cycle
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay
import pandas as pd
import sys

#Input: None
#Processing: Loads e1_positive data set, extracts- Feature names (header),
Features, and labels each as np array
#Output: Stores Feature Names, Features, and Labels as files to directory
def Process_DataSet():
    rows = []
    labels = []
    record_num = 608
    features = []
    with open("e1_positive.csv", 'r') as file:
        csvreader = csv.reader(file)
        header = next(csvreader) #list of feature types
        for index in range (0, record_num):
             header.append(header[index][:record_num]) #list of headers
        for row in csvreader:
            rows.append(row)
        for index in range (0, len(rows)):
            labels.append(rows[index][record_num]) #list of labels
        for index in range (0, len(rows)):
            features.append(rows[index][:record_num]) #list of features
        z = np.array(header)
        x = np.array (features)
        y = np.array(labels)
        i = np.argsort(np.random.random(x.shape[0]))
        x = x[i]
        y = y[i]
        np.save("e1_header.npy", z)
        np.save("e1_features.npy", x)
        np.save("e1_labels.npy", y)

#Input: None
#Processing: Creates multiple forest objects w/ 500, 1000, or 2000 trees, and 12,
24, or 48 feature samples as basis for split
#Output: Stores untrained forest objects to directory for future training
def Create_Forest_Objects():
    model_1 = Create_Forest(500, 12)
    model_1_file = 'finalized_model_1.sav'
    joblib.dump(model_1, model_1_file)
    model_2 = Create_Forest(500, 24)
    model_2_file = 'finalized_model_2.sav'
    joblib.dump(model_2, model_2_file)
    model_3 = Create_Forest(500, 48)
    model_3_file = 'finalized_model_3.sav'
    joblib.dump(model_3, model_3_file)
    model_4 = Create_Forest(1000, 12)
    model_4_file = 'finalized_model_4.sav'
    joblib.dump(model_4, model_4_file)
    model_5 = Create_Forest(1000, 24)
    model_5_file = 'finalized_model_5.sav'
    joblib.dump(model_5, model_5_file)
    model_6 = Create_Forest(1000, 48)
    model_6_file = 'finalized_model_6.sav'
    joblib.dump(model_6, model_6_file)
    model_7 = Create_Forest(2000, 12)
    model_7_file = 'finalized_model_7.sav'
    joblib.dump(model_7, model_7_file)
    model_8 = Create_Forest(2000, 24)
    model_8_file = 'finalized_model_8.sav'
    joblib.dump(model_8, model_8_file)
    model_9 = Create_Forest(2000, 48)
    model_9_file = 'finalized_model_9.sav'
    joblib.dump(model_9, model_9_file)

#Input: None
#Processing: Helper function and called by Create_Forest_Objects function
#Output: Returns a Random Forest Tree Object
def Create_Forest(forest_size, n_features):
    return RandomForestClassifier(n_estimators = forest_size, criterion='gini',
                                    max_depth=None, min_samples_split=2,
min_samples_leaf=1,
                                    min_weight_fraction_leaf=0.0, max_features=
n_features, min_impurity_decrease=0.0, bootstrap=True,
                                    oob_score=True, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None,
                                    ccp_alpha=0.0, max_samples=None)

#Input: Under generated Random Forest model selection to be loaded from directory,
the Random Forest model created by Create_Forest
```

```
#Processing: Loads the selected Random Forest
#Output: Returns the selected Random Forest
def Load_Forest(menu_selection):
    if menu_selection == 1:
        return joblib.load('finalized_model_1.sav')

    elif menu_selection == 2:
        return joblib.load('finalized_model_2.sav')

    elif menu_selection == 3:
        return joblib.load('finalized_model_3.sav')

    elif menu_selection == 4:
        return joblib.load('finalized_model_4.sav')

    elif menu_selection == 5:
        return joblib.load('finalized_model_5.sav')

    elif menu_selection == 6:
        return joblib.load('finalized_model_6.sav')

    elif menu_selection == 7:
        return joblib.load('finalized_model_7.sav')

    elif menu_selection == 8:
        return joblib.load('finalized_model_8.sav')

    elif menu_selection == 9:
        return joblib.load('finalized_model_9.sav')

#Input: Random Forest Object (clf), Feature Test Set (x_test), Label Test Set
(y_test)
#Processing: Creates labeled Confusion Matrix for Random Forest Object
#Output: PLots Confusion Matrix for Random Forest Object
def Print_Confusion_Matrix(clf, x_test, y_test):
    length = len(x_test)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_test[[index]]))
    cm = confusion_matrix(y_test, y_pred, labels=clf.classes_, normalize = None)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["Class 0",
"Class 1"])
    disp.plot(cmap=plt.cm.Blues)
    plt.show()

#Input: Trained Random Forest Object (clf), Feature Test Set (x_test), Label Test
Set (y_test)
#Processing: Creates an array (y_pred) of predicted labels from x_train, provides
y_pred and y_train to F1 function, generates OOB accuracy score for clf
#Output: Prints f1 score and OOB accuracy scores generally for clf
```

```
def Print_OOB_F1(x_test, y_test, clf):
    length = len(x_test)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_test[[index]]))
    f1 = f1_score(y_test, y_pred, labels= [0, 1], average=None)
    f1_add = f1[0] + f1[1]
    f1_average = f1_add/2
    print("F1 score generally:")
    print(f1_average)
    print("OOB score generally:")
    print(clf.oob_score_)

#Input: Feature Test Set (x_test), Label Test Set (y_test), Trained Random Forest
Object (clf)
#Processing: Creates an array (y_pred) of predicted labels from x_train, provides
y_pred and y_train to accuracy_score function
#Output: Prints Accuracy score for Trained Random Forest Object
def Print_Accuracy(x_test, y_test, clf):
    length = len(x_test)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_test[[index]]))
    accuracy = accuracy_score(y_test, y_pred, normalize=True)
    print("Accuracy is:")
    print(accuracy)

#Input: Feature Test Set (x_test), Label Test Set (y_test), Trained Random Forest
Object (clf)
#Processing: Creates an array (y_pred) of predicted labels from x_train, provides
y_pred and y_train to classfication_report function
#Output: Prints classification report with F1, Recall, and Accurcy Score
def Print_Report(x_test, y_test, clf):
    length = len(x_test)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_test[[index]]))
    target_names = ["Class 0", "Class 1"]
    print(classification_report(y_test, y_pred, target_names=target_names,
sample_weight=None, digits = 5))

#Input: Label Test Set (y_test), Feature Test Set (x_test), Trained Random Forest
Object (clf)
#Processing: Uses input to plots true positive rate over false positive rate at
different classification thresholds
#Output: Represents plot as a graph (ROC graph)
def Print_ROC(y_test, x_test, clf):
    RocCurveDisplay.from_estimator(clf, x_test, y_test)
    plt.show()
```

```
#Input: Trained Random Forest model(clf)
#Processing: Loads feature names, creates dictionary of feature names and feature
probabilities, selects top 10 feature types used by clf
#Output: Bar graph of top 10 feature types for clf, and data for top 10 features
types
def Print_MDI(clf):
    f_names = np.load("e1_header.npy")
    importances = clf.feature_importances_
    std = np.std([tree.feature_importances_ for tree in clf.estimators_], axis = 0)
    features_arr = [{key: val} for key, val in zip(f_names, std)]
    result = {}
    for features in features_arr:
        result.update(features)
    features_dict = result
    feature_rankings = {k: v for k, v in sorted(features_dict.items(), key = lambda
item:item[1], reverse = True)}
    sub_set = dict(itertools.islice(feature_rankings.items(), 10))
    plt.barh(*zip(*sub_set.items()))
    plt.show()
    print(sub_set)
    print("\n")

#Input: Label Test Set (y_test), Feature Test Set (x_test), Trained Random Forest
Object (clf)
#Processing: Clf predicts label based upon selected x_test
#Output: Predicted example classes for "0" and "1"
def Print_Test_Output(y_test, x_test, clf):
    print("Test Set #1:")
    print(x_test[1])
    print("Predicted value #1:")
    print(clf.predict(x_test[[1]]))
    print("Actual value #1:")
    print(y_test[1])
    print("Test Set #2:")
    print(x_test[6])
    print("Predicted value #2:")
    print(clf.predict(x_test[[6]]))
    print("Actual value #2:")
    print(y_test[6])

#Process_DataSet() #Run once, processes data from e1_positive data set
#Create_Forest_Objects() #Run once to create Random Forest models 1-9, stores to
directory

#Allows user to select Random Forest Models 1-9, and to train the selected model
while(True):
    model_number = int(input("Enter Model Number (1 to 9), (0 to exit):"))
    if model_number == 0:
        sys.exit("Bye Bye!!")
    else:
        x = np.load("e1_features.npy")
        y = np.load("e1_labels.npy")
        N = 611 # 30% Reserved for Test Set
        x_train = x[:N]; x_test = x[N:] #Features
        y_train = y[:N]; y_test = y[N:] #Label
        clf = Load_Forest(model_number)
        clf.fit(x_train, y_train)

#Allows user to select analysis to be performed on one of Models 1-9
    print ("Select analysis type:")
    print ("1. Enter '1' to print Confusion Matrix")
    print ("2. Enter '2' to print OOB score v. F1 score")
    print ("3. Enter '3' to print Accuracy")
    print ("4. Enter '4' to print Classification Report")
    print ("5. Enter '5' to print ROC")
    print ("6. Enter '6' to print MDI")
    print ("7. Enter '7' to print test output")
    print ("8. Enter '8' to End Program")
    menu_selection = int(input("Select Input Options: "))

    if menu_selection == 1:
        Print_Confusion_Matrix(clf, x_test, y_test)

    elif menu_selection == 2:
        Print_OOB_F1(x_test, y_test, clf)

    elif menu_selection == 3:
        Print_Accuracy(x_test, y_test, clf)

    elif menu_selection == 4:
        Print_Report(x_test, y_test, clf)

    elif menu_selection == 5:
        Print_ROC(y_test, x_test, clf)

    elif menu_selection == 6:
        Print_MDI(clf)

    elif menu_selection == 7:
        Print_Test_Output(y_test, x_test, clf)

    elif menu_selection == 8:
        sys.exit("Bye Bye!!")

    else:
        print ("Entered wrong selection, try again")
        print (" ")
```