```
"""
Author      : Madison Hobbs and Shota Yasunaga
Class       : HMC CS 158
Date        : 2018 Mar 05
Description : Bagging with Digits Dataset
              This code was adapted from course material by Jenna Wiens (UMich)
"""

# python libraries
import collections

# numpy libraries
import numpy as np

# matplotlib libraries
import matplotlib.pyplot as plt

# scikit-learn libraries
from sklearn.datasets import load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics


######################################################################
# bagging functions
######################################################################

def bagging_ensemble(X_train, y_train, X_test, y_test, max_features=None, num_clf=10
) :
    """
    Compute performance of bagging ensemble classifier.

    Parameters
    --------------------
        X_train      -- numpy array of shape (n_train,d), training features
        y_train      -- numpy array of shape (n_train,),  training targets
        X_test       -- numpy array of shape (n_test,d),  test features
        y_test       -- numpy array of shape (n_test,),   test targets
        max_features -- int, number of features to consider when looking for best sp
lit
        num_clf      -- int, number of decision tree classifiers in bagging ensemble

    Returns
    --------------------
        accuracy     -- float, accuracy of bagging ensemble classifier on test data
    """
    base_clf = DecisionTreeClassifier(criterion='entropy', max_features=max_features
)
    clf = BaggingClassifier(base_clf, n_estimators=num_clf)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    return metrics.accuracy_score(y_test, y_pred)


def random_forest(X_train, y_train, X_test, y_test, max_features, num_clf=10,
                  bagging=bagging_ensemble) :
    """
    Wrapper around bagging_ensemble to use feature-limited decision trees.

    Additional Parameters
    --------------------
        bagging      -- bagging_ensemble or bagging_ensemble2
    """
    return bagging(X_train, y_train, X_test, y_test,
                    max_features=max_features, num_clf=num_clf)


def bagging_ensemble2(X_train, y_train, X_test, y_test, max_features=None, num_clf=1
0) :
    """
    Compute performance of bagging ensemble classifier.
    You are allowed to use DecisionTreeClassifier but NOT BaggingClassifier.

    Details
    - Train num_clf base classifiers using bootstrap samples from X_train and y_train.
      Use DecisionTreeClassifier with information gain as base classifier.
      Hints: Use np.random.choice(...) for bootstrap samples.
             Make sure to use same indices from X_train and y_train.
    - Predict using X_test and y_test.
      For each base classifier, track predictions on X_test.
      Make ensemble prediction using using majority vote.
    - Return accuracy compared to y_test.

    Same parameters and return values as bagging_ensemble(...)
    """

    n_train, d = X_train.shape

    ### ========== TODO : START ========== ###
    # extra credit: implement bagging ensemble (see details above)

    return 0.0
    ### ========== TODO : START ========== ###


######################################################################
# plotting functions
######################################################################

def plot_scores(max_features, bagging_scores, random_forest_scores) :
    """
    Plot values in random_forest_scores and bagging_scores.
    (The scores should use the same set of 100 different train and test set splits.)

    Parameters
    --------------------
        max_features         -- list, number of features considered when looking for best split
        bagging_scores       -- list, accuracies for bagging ensemble classifier using DTs
        random_forest_scores -- list, accuracies for random forest classifier
    """

    plt.figure()
    plt.plot(max_features, bagging_scores, '--', label='bagging')
    plt.plot(max_features, random_forest_scores, '--', label='random forest')
    plt.xlabel('max features considered per split')
    plt.ylabel('accuracy')
    plt.legend(loc='upper right')
    plt.show()


def plot_histograms(bagging_scores, random_forest_scores):
    """
    Plot histograms of values in random_forest_scores and bagging_scores.
    (The scores should use the same set of 100 different train and test set splits.)

    Parameters
    --------------------
        bagging_scores       -- list, accuracies for bagging ensemble classifier using DTs
        random_forest_scores -- list, accuracies for random forest classifier
    """

    bins = np.linspace(0.8, 1.0, 100)
    plt.figure()
    plt.hist(bagging_scores, bins, alpha=0.5, label='bagging')
    plt.hist(random_forest_scores, bins, alpha=0.5, label='random forest')
    plt.xlabel('accuracy')
    plt.ylabel('frequency')
    plt.legend(loc='upper left')
    plt.show()
```

```
######################################################################
# main
######################################################################

def main():
    np.random.seed(1234)

    # load digits dataset
    digits = load_digits(4)
    X = digits.data
    y = digits.target

    # evaluation parameters
    num_trials = 100

    # sklearn or home-grown bagging ensemble
    bagging = bagging_ensemble

    #========================================
    # vary number of features

    # calculate accuracy of bagging ensemble and random forest
    #   for 100 random training and test set splits
    # make sure to use same splits to enable proper comparison
    '''max_features_vector = range(1,65,2)
    bagging_scores = []
    random_forest_scores = collections.defaultdict(list)
    for i in range(num_trials):
        print i
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        bagging_scores.append(bagging(X_train, y_train, X_test, y_test))
        for m in max_features_vector :
            random_forest_scores[m].append(random_forest(X_train, y_train, X_test, y
_test, m,
                                                         bagging=bagging))

    # analyze how performance of bagging and random forest changes with m
    bagging_results = []
    random_forest_results = []
    for m in max_features_vector :
        bagging_results.append(np.median(np.array(bagging_scores)))
#        print m, np.median(np.array(random_forest_scores[m]))
        random_forest_results.append(np.median(np.array(random_forest_scores[m])))
    plot_scores(max_features_vector, bagging_results, random_forest_results)

    #========================================
    # plot histograms of performances for max_features=8
    bagging_scores = []
    random_forest_scores = []
    for i in range(num_trials) :
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        bagging_scores.append(bagging(X_train, y_train, X_test, y_test))
        random_forest_scores.append(random_forest(X_train, y_train, X_test, y_test,
8,
                                    bagging=bagging))
    plot_histograms(bagging_scores, random_forest_scores)
    '''
    ### ========== TODO : START ========== ###
    # part b: determine pixel importance
    clf = RandomForestClassifier(criterion="entropy")
    clf.fit(X, y)
    importance = clf.feature_importances_
    importance = importance.reshape(digits.images[0].shape)

    plt.matshow(importance, cmap = plt.cm.hot)
    plt.title("Pixel importances with forests of trees")
    plt.show()

    ### ========== TODO : END ========== ###
```

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