APPENDIX: CODE

```
"""
EECS 445 - Introduction to Machine Learning
Winter 2019
Homework 2, Ensemble Methods
Skeleton Code
"""

import random
import numpy as np
import matplotlib.pyplot as plt

from collections import Counter
from sklearn import metrics, utils
from sklearn.datasets import fetch_mldata
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split


def load_mnist(classes):
    """
    Load MNIST dataset for classes
    Every 25th sample is used to reduce computational resources
    Input:
        classes : list of ints
    Returns:
        X : np.array (num_samples, num_features)
        y : np.array (num_samples)
    """
    print('Fetching MNIST data...')
    mnist = fetch_mldata('MNIST original')
    X_all = np.array(mnist.data)[::25]
    y_all = np.array(mnist.target)[::25]
    desired_idx = np.isin(y_all, classes)
    return X_all[desired_idx], y_all[desired_idx]


def get_avg_performance(X, y, m_vals, n_splits=50):
    """
    Compare the average performance of bagging and random forest across 50
    random splits of X and y
    Input:
        X : np.array (num_samples, num_features)
        y : np.array (num_samples)
        m_vals: list - list of values for m
        n_splits: int - number of random splits
    Returns:
        bag_results : np.array (len(m_vals)) - estimate of bagging performance
        rf_results : np.array (len(m_vals)) - estimate of random forest performance
    """
    print('Getting bagging and random forest scores...')
    rf_results = []
    bag_results = []
    for m in m_vals:
        print('m = {}'.format(m))
        bagging_scores = []
        random_forest_scores = []
        for i in range(n_splits):
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
            random_forest_scores.append(random_forest(X_train, y_train, X_test, y_test, m))
            bagging_scores.append(bagging_ensemble(X_train, y_train, X_test, y_test))
        rf_results.append(np.median(np.array(random_forest_scores)))
        bag_results.append(np.median(np.array(bagging_scores)))
    return bag_results, rf_results


def plot_data(bagging_scores, random_forest_scores, m_vals):
    """
    Plot bagging and random forest accuracies
    Input:
        bagging_scores : np.array - array containing accuracies for bagging ensemble classifiers
        random_forest_scores : np.array - array containing accuracies for random forest classifiers
    """
    plt.figure()
    plt.plot(list(m_vals), bagging_scores, '--', label='bagging')
    plt.plot(list(m_vals), random_forest_scores, '--', label='random forest')
    plt.xlabel('m')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.savefig('ensemble.png', dpi=300)
    plt.show()


def random_forest(X_train, y_train, X_test, y_test, m, n_clf=10):
    """
    Returns accuracy on the test set X_test with corresponding labels y_test
    using a random forest classifier with n_clf decision trees trained with
    training examples X_train and training labels y_train.
    Input:
        X_train : np.array (n_train, d) - array of training feature vectors
        y_train : np.array (n_train) - array of labels corresponding to X_train samples
        X_test : np.array (n_test,d) - array of testing feature vectors
        y_test : np.array (n_test) - array of labels corresponding to X_test samples
        m : int - number of features to consider when splitting
        n_clf : int - number of decision tree classifiers in the random forest, default is 10
    Returns:
        accuracy : float - accuracy of random forest classifier on X_test samples
    """
    # TODO: Implement this function

    y_predict = np.zeros((10,X_test.shape[0]))
    boot_size = X_train.shape[0]
    for i in range(10):
        X_boot, y_boot = utils.resample(X_train, y_train, n_samples = boot_size)
        clf = DecisionTreeClassifier(criterion = 'entropy', max_features = m)
        clf.fit(X_boot, y_boot)
        y_pred = clf.predict(X_test)
        y_predict[i] = y_pred

    y_pred = []
    for i in range(X_test.shape[0]):
        y_pred = np.append(y_pred,Counter(y_predict[:,i]).most_common(1)[0][0])
    return metrics.accuracy_score(y_test, y_pred)


def bagging_ensemble(X_train, y_train, X_test, y_test, n_clf=10):
    """
    Returns accuracy on the test set X_test with corresponding labels y_test
    using a bagging ensemble classifier with n_clf decision trees trained with
    training examples X_train and training labels y_train.
    Input:
        X_train : np.array (n_train, d) - array of training feature vectors
        y_train : np.array (n_train) - array of labels corresponding to X_train samples
        X_test : np.array (n_test,d) - array of testing feature vectors
        y_test : np.array (n_test) - array of labels corresponding to X_test samples
        n_clf : int - number of decision tree classifiers in the random forest, default is 10
    Returns:
        accuracy : float - accuracy of random forest classifier on X_test samples
    """
    # TODO: Implement this function

    y_predict = np.zeros((10,X_test.shape[0]))
    boot_size = X_train.shape[0]
    for i in range(10):
        X_boot, y_boot = utils.resample(X_train, y_train, n_samples = boot_size)
        clf = DecisionTreeClassifier(criterion = 'entropy')
        clf.fit(X_boot, y_boot)
        y_pred = clf.predict(X_test)
        y_predict[i] = y_pred

    y_pred = []
    for i in range(X_test.shape[0]):
        y_pred = np.append(y_pred,Counter(y_predict[:,i]).most_common(1)[0][0])


    return metrics.accuracy_score(y_test, y_pred)


def main():
    """
    Analyze how the performance of bagging and random forest changes with m.
    """
    X, y = load_mnist([1,2,3,4])
    # Plot accuracies
    m_vals = [1] + list(range(56, 785, 56))
    bagging_scores, random_forest_scores = get_avg_performance(X, y, m_vals)
    plot_data(bagging_scores, random_forest_scores, m_vals)


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