```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import StratifiedKFold, train_test_split
from sklearn import metrics

def cv_error(clf, X, y, k=5):
    """
        Splits the data, X and y, into k-folds. Trains a classifier on K-1
            folds and
        testing on the remaining fold. Calculates the cross validation train
            and test
        error for classifier clf by averaging the performance across folds.
        Input:
          clf- an instance of any classifier
          X- (n,d) array of feature vectors, where n is # of examples
             and d is # of features
          y- (n,) array of binary labels {1,-1}
          k- int, # of folds (default=5)

        Returns: average train and test error across the k folds as np.float64
    """
    skf = StratifiedKFold(y, k)
    train_scores, test_scores = [], []
    for train, test in skf:
        X_train, X_test, y_train, y_test = X[train], X[test], y[train], y[test]
        clf.fit(X_train, y_train)
        train_scores.append(clf.score(X_train, y_train))
        test_scores.append(clf.score(X_test, y_test))

    return 1 - np.array(train_scores).mean(), 1 - np.array(test_scores).mean()


def plot_error(X, y):
    '''
    Plots the variation of 5-fold cross-validation error w.r.t. maximum
    depth of the decision tree
        X- (n, d) array of feature vectors, where n is # of examples
             and d is # of features
        y- (n, ) array of labels corresponding to X
    '''

    # ---------- make your implementation here---------------
    train_error = []
    test_error = []
    max_depth = list(range(1, 21))
    for i in range(1,21):
        clf = DecisionTreeClassifier(criterion="entropy", max_depth=i)
        result = cv_error(clf, X, y, 5)
        train_error.append(result[0])
        test_error.append(result[1])
    plt.plot(max_depth, train_error, label="train error")
    plt.plot(max_depth, test_error, label="test error")
    plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)
```

```
    plt.ylabel('errors')
    plt.xlabel('max depth')
    plt.savefig('max_depth_vs_error.png')
    #plt.show()
    # ------------------------------------------------------------

def majority_vote(pred_list):
    '''
    Given a list of m (n, ) arrays, each (n, ) array containing the predictions
        for
    same set of n examples (using different classifiers), return a (n, ) array
    containing majority vote prediction of the m (n, ) arrays
    Input:
        pred_list- a list of m (n, ) arays
    Output:
        y_pred- (n, ) array containing majority vote prediction using pred_list
    '''
    y_pred = []
    for i in range(len(pred_list[0])):
        lst = [row[i] for row in pred_list]
        y_pred.append(max(set(lst), key = lst.count))

    return y_pred


def bagging_ensemble(X_train, y_train, X_test, y_test, m = None, 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- (n_train, d) array of training feature vectors, where n_train
            is # of examples and d is # of features
        y_train- (n_train, ) array of labels corresponding to X_train
        X_test- (n_test, d) array of testing feature vectors, where n_test is
            # of examples and d is # of features
        y_test- (n_test, ) array of labels corresponding to X_test
        m - int, # of features to consider when looking for the best split
        n_clf- # of decision tree classifiers in the bagging ensemble, default
            value of n_clf is 10
    Output:
        Accuracy of the bagging ensemble classifier on X_test
    '''

    # ---------- make your implementation here---------------
    bagging_ratio = 0.6
    total_size = y_train.shape[0]
    bagging_size = int(bagging_ratio * total_size)
    bagsX = []
    bagsY = []
    for i in range(0, n_clf):
        tempx = []
        tempy = []
        for j in range(0, bagging_size):
            rand = np.random.randint(0, total_size)
            tempx.append(X_train[rand, :])
```

```
            tempy.append(y_train[rand])
        bagsX.append(tempx)
        bagsY.append(tempy)
    clfs = []
    accuracy = 0
    for i in range(0, n_clf):
        clfs.append(DecisionTreeClassifier(max_features=m));
    for i in range(0, n_clf):
        clfs[i].fit(bagsX[i], bagsY[i])
    preds = []
    for i in range(0, n_clf):
        temp = clfs[i].predict(X_test)
        preds.append(temp)
    y_pred = majority_vote(preds)
    count = 0
    for i in range(0, y_test.shape[0]):
        if y_pred[i] == y_test[i]:
            count = count + 1
    accuracy = count / y_test.shape[0]
    return accuracy
    # ------------------------------------------------------


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- (n_train, d) array of training feature vectors, where n_train
            is # of examples and d is # of features
        y_train- (n_train, ) array of labels corresponding to X_train
        X_test- (n_test, d) array of testing feature vectors, where n_test is
            # of examples and d is # of features
        y_test- (n_test, ) array of labels corresponding to X_test
        n_clf- # decision tree classifiers in the random forest, default
            value of n_clf is 10
    Output:
        Accuracy of the random forest classifier on X_test
    '''

    # ---------- make your implementation here---------------
    accuracy = bagging_ensemble(X_train, y_train, X_test, y_test, m, n_clf)
    return accuracy
    # ------------------------------------------------------


def plot_histograms(random_forest_scores, bagging_scores):
    '''
    Plots histogram of values in random_forest_scores and bagging_scores
    overlayed on top of each other
    Input:
        random_forest_scores- a list containing accuracy values for random
            forest classifier
        for 100 different train and test set splits
        bagging_scores- a list containing accuracy values for bagging ensemble
```

```
            classifier
        using decision trees for the same 100 different train and test set
            splits
        as random_forest_scores
    '''
    bins = np.linspace(0.8, 1.0, 100)
    plt.figure()
    plt.hist(random_forest_scores, bins, alpha=0.5, label='random forest')
    plt.hist(bagging_scores, bins, alpha=0.5, label='bagging')
    plt.xlabel('Accuracy')
    plt.ylabel('Frequency')
    plt.legend(loc='upper left')
    plt.savefig('histogram.png')
    print("all done")
    #plt.show()


def q1a():
    # Load Iris dataset
    iris = load_iris()
    X, y = iris.data[:,:2], iris.target
    # PLot cross-validation error vs max_depth
    plot_error(X, y)


def q1b():
    # Load digits dataset
    digits = load_digits(4)
    X, y = digits.data, digits.target

    # Calculate accuracy of bagging ensemble and random forest for 100 random
        train/test splits
    # Analyze how the performance of bagging & random forest changes with m
    results1, results2 = [], []
    for j in range(0, 64, 2):
        print (j / 64 * 100, "%")
        bagging_scores, random_forest_scores = [], []
        for i in range(100):
            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, j+1))
            bagging_scores.append(bagging_ensemble(X_train, y_train, X_test,
                y_test))
        results1.append(np.median(np.array(random_forest_scores)))
        results2.append(np.median(np.array(bagging_scores)))

    plt.figure()
    plt.plot(range(0, 64, 2), results1, '--', label = 'random forest')
    plt.plot(range(0, 64, 2), results2, '--', label = 'bagging')
    plt.xlabel('m')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.savefig('second_plot.png')
    #plt.show()
```

```
        print("second plot finished, on to histogram")

        #Plot histogram of performances for m = 8
        bagging_scores, random_forest_scores = [], []
        for i in range(100):
            print(i, "%")
            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,8))
            bagging_scores.append(bagging_ensemble(X_train, y_train, X_test, y_test
                ))
        plot_histograms(random_forest_scores, bagging_scores)

    if __name__ == '__main__':
        q1a()
        q1b()
```