```
"""
Author      : Jackson Crewe and Matt Guillory
Assignment  : 2
Class       : HMC CS 158
Date        : 2017 Aug 02
Description : Titanic
"""

# Use only the provided packages!
import math
import csv
from util import *

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

######################################################################
# classes
######################################################################

class Classifier(object) :
    """
    Classifier interface.
    """

    def fit(self, X, y):
        raise NotImplementedError()

    def predict(self, X):
        raise NotImplementedError()


class MajorityVoteClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that always predicts the majority class.

        Attributes
        --------------------
            prediction_ -- majority class
        """
        self.prediction_ = None

    def fit(self, X, y) :
        """
        Build a majority vote classifier from the training set (X, y).

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            y    -- numpy array of shape (n,), target classes

        Returns
        --------------------
            self -- an instance of self
        """
        vals, counts = np.unique(y, return_counts=True)
        majority_val, majority_count = max(zip(vals, counts), key=lambda (val, count
): count)
        self.prediction_ = majority_val
        return self

    def predict(self, X) :
        """
        Predict class values.

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples

        Returns
```

```
        --------------------
            y    -- numpy array of shape (n,), predicted classes
        """
        if self.prediction_ is None :
            raise Exception("Classifier not initialized. Perform a fit first.")

        n,d = X.shape
        y = [self.prediction_] * n
        return y


class RandomClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that predicts according to the distribution of the classes.

        Attributes
        --------------------
            probabilities_ -- class distribution dict (key = class, val = probabilit
y of class)
        """
        self.probabilities_ = None

    def fit(self, X, y) :
        """
        Build a random classifier from the training set (X, y).

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            y    -- numpy array of shape (n,), target classes

        Returns
        --------------------
            self -- an instance of self
        """
        # Generate the counts and probabilities for the majority and minority values
        vals, counts = np.unique(y, return_counts=True)
        majority_val, majority_count = max(zip(vals, counts), key=lambda (val, count
): count)
        minority_val, minority_count = min(zip(vals, counts), key=lambda (val, count
): count)
        total_count = majority_count + minority_count
        majority_probability = float(majority_count) / float(total_count)
        minority_probability = float(minority_count) / float(total_count)

        # Generate dictionary using the values calculated above
        self.probabilities_ = {majority_val : majority_probability, minority_val : m
inority_probability}
        return self

    def predict(self, X, seed=1234) :
        """
        Predict class values.

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            seed -- integer, random seed

        Returns
        --------------------
            y    -- numpy array of shape (n,), predicted classes
        """
        if self.probabilities_ is None :
            raise Exception("Classifier not initialized. Perform a fit first.")
        np.random.seed(seed)

         # np.random.choice assigns the keys into the array at the probability speci
fied in its last parameter
        y = np.random.choice(self.probabilities_.keys(), len(X), True, self.probabil
ities_.values())
```

```
        return y


        return y


##########################################################################
# functions
##########################################################################

def error(clf, X, y, ntrials=100, test_size=0.2) :
    """
    Computes the classifier error over a random split of the data,
    averaged over ntrials runs.

    Parameters
    --------------------
        clf         -- classifier
        X           -- numpy array of shape (n,d), features values
        y           -- numpy array of shape (n,), target classes
        ntrials     -- integer, number of trials
        test_size   -- float (between 0.0 and 1.0) or int,
                       if float, the proportion of the dataset to include in the tes
t split
                       if int, the absolute number of test samples

    Returns
    --------------------
        train_error -- float, training error
        test_error  -- float, test error
    """

    train_error = 0
    test_error = 0
    for i in range(1,ntrials):
        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                test_size=test_size, random_state=ntrial
s)
        clf.fit(X_train, y_train)               # create model based on training dat
a
        y_pred_train = clf.predict(X_train)     # take the classifier and run it on
the training data
        y_pred_test = clf.predict(X_test)       # take the classifer and run it on t
he testing data

        train_error += 1 - metrics.accuracy_score(y_train, y_pred_train, normalize=T
rue)
        test_error += 1 - metrics.accuracy_score(y_test, y_pred_test, normalize=True
)

    train_error = train_error / ntrials
    test_error = test_error / ntrials

    return train_error, test_error


def write_predictions(y_pred, filename, yname=None) :
    """Write out predictions to csv file."""
    out = open(filename, 'wb')
    f = csv.writer(out)
    if yname :
        f.writerow([yname])
    f.writerows(zip(y_pred))
    out.close()

def plot_depth(X, y, test_error_majority, test_error_random):
    """
    Plots average training error and test error against the depth limit. Also
    includes the average test error for the baseline classifers (MajorityVoteClassif
ier
    and RandomClassifier).

    Parameters
```

```
    --------------------
    X                   -- numpy array of shape (n,d), features values
    y                   -- numpy array of shape (n,), target classes
    test_error_majority -- MajorityVoteClassifier test error
    test_error_random   -- RandomClassifier test error
    """

    depth_points = np.arange(20)
    test_error_majority_points = np.full(20, test_error_majority)
    test_error_random_points = np.full(20, test_error_random)
    test_error_tree_points = np.ones(20)
    train_error_tree_points = np.ones(20)

    min_error = 1
    min_tree_depth = 0

    for i in range(1,21):
        clf = DecisionTreeClassifier(criterion='entropy', max_depth=i)
        train_error_tree, test_error_tree = error(clf, X, y)
        test_error_tree_points[i-1] = test_error_tree
        train_error_tree_points[i-1] = train_error_tree
        if test_error_tree < min_error:
            min_error = test_error_tree
            min_tree_depth = i

    plt.plot(depth_points, test_error_majority_points, 'r--', label='Majority')
    plt.plot(depth_points, test_error_random_points, 'm--', label='Random')
    plt.plot(depth_points, test_error_tree_points, 'g^',label='Tree test')
    plt.plot(depth_points, train_error_tree_points, 'cP', label='Tree train')

    plt.xlabel('Tree Depth')
    plt.ylabel('Error')
    plt.legend()
    # print min_error
    # print min_tree_depth
    plt.show()

def plot_learning_curves(X, y, test_error_majority, test_error_random):
    """
    Plots average training error and test error against the percentage of data used
as traing data.
    Also includes the average test error for the baseline classifers (MajorityVoteCl
assifier
    and RandomClassifier).

    Parameters
    --------------------
    X                   -- numpy array of shape (n,d), features values
    y                   -- numpy array of shape (n,), target classes
    test_error_majority -- MajorityVoteClassifier test error
    test_error_random   -- RandomClassifier test error
    """
    learning_points = np.arange(.05, .95, .05)
    test_error_majority_points = np.full(18, test_error_majority)
    test_error_random_points = np.full(18, test_error_random)
    test_error_learning_points = np.ones(18)
    train_error_learning_points = np.ones(18)

    for i in range(0, 18):
        clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
        test_size = 1-learning_points[i]
        train_error_learning, test_error_learning = error(clf, X, y, test_size=test_
size)
        test_error_learning_points[i] = test_error_learning
        train_error_learning_points[i] = train_error_learning

    plt.plot(learning_points, test_error_majority_points, 'r--', label='Majority')
    plt.plot(learning_points, test_error_random_points, 'm--', label='Random')
    plt.plot(learning_points, test_error_learning_points, 'g^',label='Tree test')
    plt.plot(learning_points, train_error_learning_points, 'cP', label='Tree train')

    plt.xlabel('Percentage of Training Data')
    plt.ylabel('Error')
```

```
    plt.legend()
    plt.show()

def frange(x, y, jump):
  while x < y:
    yield x
    x += jump

def make_contest_predictions(X, y):
    """
    Tests different hyperparameters for contest using input dataset

    Parameters
    --------------------
    X                   -- numpy array of shape (n,d), features values
    y                   -- numpy array of shape (n,), target classes
    """
    min_error = 1
    best_depth = 0
    best_min_impurity_decrease = 0
    best_samp_split = 0

    for i in range(2,20):
        for j in range(1,10):
            for k in np.arange(.05, .30, .05):
                print k

                train_error_tree, test_error_tree = error(clf, X, y)
                if (test_error_tree < min_error):
                    min_error = test_error_tree
                    best_depth = j
                    best_min_impurity_decrease = k
                    best_samp_split = i
                print """The tree classifier average training cross validation error
                is {0:.3f} and the average testing cross validation error is {1:.3f}
.
                The max depth is {2}, the min impurity decrease is {3}, and the min
samples split is {4}""".format(train_error_tree, test_error_tree, j, k, i) + '\n'

    print "The best hyperparameters are: best error: {3}, best depth: {0}, best imp:
 {1}, best samp: {2}".format(best_depth, best_min_impurity_decrease, best_samp_split
, min_error)


######################################################################
# main
######################################################################

def main():
    # load Titanic dataset
    titanic = load_data("titanic_train.csv", header=1, predict_col=0)
    X = titanic.X; Xnames = titanic.Xnames
    y = titanic.y; yname = titanic.yname
    n,d = X.shape  # n = number of examples, d =  number of features



    #========================================
    # train Majority Vote classifier on data
    print 'Classifying using Majority Vote...'
    clf = MajorityVoteClassifier() # create MajorityVote classifier, which includes
all model parameters
    clf.fit(X, y)                  # fit training data using the classifier
    y_pred = clf.predict(X)        # take the classifier and run it on the training
data
    train_error = 1 - metrics.accuracy_score(y, y_pred, normalize=True)
    print '\t-- training error: %.3f' % train_error



    print 'Classifying using Decision Tree...'
    dtc = DecisionTreeClassifier(criterion='entropy')
    dtc.fit(X,y)
```

```
    y_pred = dtc.predict(X)
    train_error = 1 - metrics.accuracy_score(y,y_pred, normalize=True)
    print '\t-- training error: %.3f' % train_error



    # note: uncomment out the following lines to output the Decision Tree graph
    """
    # save the classifier -- requires GraphViz and pydot
    import StringIO, pydot
    from sklearn import tree
    dot_data = StringIO.StringIO()
    tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=Xnames,
                         class_names=["Died", "Survived"])
    graph = pydot.graph_from_dot_data(dot_data.getvalue())
    graph.write_pdf("dtree.pdf")
    """

    print 'Investigating various classifiers...'
    clf1 = MajorityVoteClassifier();
    train_error_majority, test_error_majority = error(clf1, X, y)

    clf2 = RandomClassifier();
    train_error_random, test_error_random = error(clf2, X, y)

    clf3 = DecisionTreeClassifier(criterion='entropy')
    train_error_tree, test_error_tree = error(clf3, X, y)

    print """The majority vote classifier average training cross validation error
    is {0:.3f} and the average testing cross validation error is {1:.3f}""".format(t
rain_error_majority, test_error_majority)

    print """The random classifier average training cross validation error
    is {0:.3f} and the average testing cross validation error is {1:.3f}""".format(t
rain_error_random, test_error_random)

    print """The tree classifier average training cross validation error
    is {0:.3f} and the average testing cross validation error is {1:.3f}""".format(t
rain_error_tree, test_error_tree)


    print 'Investigating depths...'
    plot_depth(X, y, test_error_majority, test_error_random)


    # part d: investigate decision tree classifier with various training set sizes
    print 'Investigating training set sizes...'
    plot_learning_curves(X, y, test_error_majority, test_error_random)


    # Contest
    # uncomment write_predictions and change the filename

    # evaluate on test data
    titanic_test = load_data("titanic_test.csv", header=1, predict_col=None)
    X_test = titanic_test.X
    clf = DecisionTreeClassifier(criterion='entropy', max_depth = 3,
                                 min_impurity_decrease = 0.15, min_samples_split = 10
)
    clf.fit(X,y)
    y_pred = clf.predict(X_test)   # take the trained classifier and run it on the t
est data
    write_predictions(y_pred, "../data/jcrewe_mguillory_titanic.csv", titanic.yname)



    print 'Done'


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