```
"""
Author      : Jackson Crewe & Matt Guillory
Class       : HMC CS 158
Date        : 2018 Feb 14
Description : Perceptron vs Logistic Regression on a Phoneme Dataset
"""

# utilities
from util import *

# scipy libraries
from scipy import stats

# scikit-learn libraries
from sklearn import preprocessing
from sklearn import metrics
from sklearn import model_selection
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import Perceptron, LogisticRegression

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

def cv_performance(clf, train_data, kfs) :
    """
    Determine classifier performance across multiple trials using cross-validation

    Parameters
    --------------------
        clf        -- classifier
        train_data -- Data, training data
        kfs        -- array of size n_trials
                      each element is one fold from model_selection.KFold

    Returns
    --------------------
        scores     -- numpy array of shape (n_trials, n_fold)
                      each element is the (accuracy) score of one fold in one trial
    """

    n_trials = len(kfs)
    n_folds = kfs[0].n_splits
    scores = np.zeros((n_trials, n_folds))

    ### ========== TODO : START ========== ###
    for i in range(n_trials):
        scores[i] = cv_performance_one_trial(clf, train_data, kfs[i])

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

    return scores


def cv_performance_one_trial(clf, train_data, kf) :
    """
    Compute classifier performance across multiple folds using cross-validation

    Parameters
    --------------------
        clf        -- classifier
        train_data -- Data, training data
        kf         -- model_selection.KFold

    Returns
    --------------------
        scores     -- numpy array of shape (n_fold, )
                      each element is the (accuracy) score of one fold
    """

    scores = np.zeros(kf.n_splits)

    ### ========== TODO : START ========== ###
    index = 0
    # compute different splits of the data
    for train_index, test_index in kf.split(train_data.X, train_data.y):
        X_train, X_test = train_data.X[train_index], train_data.X[test_index]
        y_train, y_test = train_data.y[train_index], train_data.y[test_index]
        clf.fit(X_train, y_train)
        scores[index] = clf.score(X_test, y_test)
        index += 1

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

    return scores

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

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

    #========================================
    # load data
    train_data = load_data("phoneme_train.csv")

    ### ========== TODO : START ========== ###
    clf = Perceptron(max_iter = 1000)
    clf.fit(train_data.X, train_data.y)
    print 'score = %d' % clf.score(train_data.X, train_data.y)

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

    ### ========== TODO : START ========== ###

    # create model_selection.KFold for computing classifier performance
    kfs = [model_selection.KFold(n_splits=10, shuffle=True, random_state=i) for i in
 range(10)]

    # computes classifier performance for each classifier
    clf1 = Perceptron(max_iter = 1000)
    perceptron_scores = cv_performance(clf1, train_data, kfs)
    perceptron_scores = perceptron_scores.flatten()
    print perceptron_scores

    # create dummy classifier to predict the most frequent label
    clf2 = DummyClassifier(strategy = 'most_frequent')
    dummy_scores = cv_performance(clf2, train_data, kfs)
    dummy_scores = dummy_scores.flatten()
    print dummy_scores

    # create logistical regression model with high C value to limit regularization
    clf3 = LogisticRegression(C = 1e5)
    logistic_scores = cv_performance(clf3, train_data, kfs)
    logistic_scores = logistic_scores.flatten()
    print logistic_scores

    perceptron_mean = np.mean(perceptron_scores, dtype=np.float64)
    perceptron_sdev = np.std(perceptron_scores, dtype=np.float64)

    dummy_mean = np.mean(dummy_scores, dtype=np.float64)
    dummy_sdev = np.std(dummy_scores, dtype=np.float64)

    logistic_mean = np.mean(logistic_scores, dtype=np.float64)
    logistic_sdev = np.std(logistic_scores, dtype=np.float64)

    t_perc_dum, p_perc_dum = stats.ttest_rel(perceptron_scores, dummy_scores)
    t_perc_log, p_perc_log = stats.ttest_rel(perceptron_scores, logistic_scores)
    t_dum_log, p_dum_log = stats.ttest_rel(dummy_scores, logistic_scores)

    print 'the perceptron mean is: %03.3f. The standard deviation is: %03.3f' % (per
ceptron_mean, perceptron_sdev)
    print 'the dummy mean is: %03.3f. The standard deviation is: %03.3f' % (dummy_me
an, dummy_sdev)
    print 'the logistic mean is: %03.3f. The standard deviation is: %03.3f' % (logis
tic_mean, logistic_sdev)
    print 'the p-value for perceptron vs dummy is: %f' % p_perc_dum
    print 'the p-value for perceptron vs logistic is: %f' % p_perc_log
    print 'the p-value for dummy vs logistic is: %f' % p_dum_log
    print 'the t-statistic for perceptron vs dummy is: %f' % t_perc_dum
    print 'the t-statistic for perceptron vs logistic is: %f' % t_perc_log
    print 'the t-statistic for dummy vs logistic is: %f' % t_dum_log

    # scales the data
    X_scaled = preprocessing.scale(train_data.X)
    train_scaled = Data(X_scaled, train_data.y)

    # computes classifier performance for each classifier
    clf1 = Perceptron(max_iter = 1000)
    perceptron_scores_scaled = cv_performance(clf1, train_scaled, kfs)
    perceptron_scores_scaled = perceptron_scores_scaled.flatten()
    print perceptron_scores_scaled

    clf2 = DummyClassifier(strategy = 'most_frequent')
    dummy_scores_scaled = cv_performance(clf2, train_scaled, kfs)
    dummy_scores_scaled = dummy_scores_scaled.flatten()
    print dummy_scores_scaled

    clf3 = LogisticRegression(C = 1e5)
    logistic_scores_scaled = cv_performance(clf3, train_scaled, kfs)
    logistic_scores_scaled = logistic_scores_scaled.flatten()
    print logistic_scores_scaled

    perceptron_mean_scaled = np.mean(perceptron_scores_scaled, dtype=np.float64)
    perceptron_sdev_scaled = np.std(perceptron_scores_scaled, dtype=np.float64)

    dummy_mean_scaled = np.mean(dummy_scores_scaled, dtype=np.float64)
    dummy_sdev_scaled = np.std(dummy_scores_scaled, dtype=np.float64)

    logistic_mean_scaled = np.mean(logistic_scores_scaled, dtype=np.float64)
    logistic_sdev_scaled = np.std(logistic_scores_scaled, dtype=np.float64)

    t_perc_dum_s, p_perc_dum_s = stats.ttest_rel(perceptron_scores_scaled, dummy_sco
res_scaled)
    t_perc_log_s, p_perc_log_s = stats.ttest_rel(perceptron_scores_scaled, logistic_
scores_scaled)
    t_dum_log_s, p_dum_log_s = stats.ttest_rel(dummy_scores_scaled, logistic_scores_
scaled)

    print 'the perceptron mean_scaled is: %03.3f. The standard deviation is: %03.3f'
 % (perceptron_mean_scaled, perceptron_sdev_scaled)
    print 'the dummy mean_scaled is: %03.3f. The standard deviation is: %03.3f' % (d
ummy_mean_scaled, dummy_sdev_scaled)
    print 'the logistic mean_scaled is: %03.3f. The standard deviation is: %03.3f' %
 (logistic_mean_scaled, logistic_sdev_scaled)
    print 'the p-value for perceptron vs dummy is: %f' % p_perc_dum_s
    print 'the p-value for perceptron vs logistic is: %f' % p_perc_log_s
    print 'the p-value for dummy vs logistic is: %f' % p_dum_log_s
    print 'the t-statistic for perceptron vs dummy is: %f' % t_perc_dum_s
    print 'the t-statistic for perceptron vs logistic is: %f' % t_perc_log_s
    print 'the t-statistic for dummy vs logistic is: %f' % t_dum_log_s


    # create bar chart to visualize results
    N = 2
    perceptron_means = (perceptron_mean, perceptron_mean_scaled)
    perceptron_sdevs = (perceptron_sdev, perceptron_sdev_scaled)

    logistic_means = (logistic_mean, logistic_mean_scaled)
    logistic_sdevs = (logistic_sdev, logistic_sdev_scaled)

    ind = np.arange(N)
    width = 0.35

    fig, ax = plt.subplots()
    rects1 = ax.bar(ind, perceptron_means, width, color='r', yerr=perceptron_sdevs)
    rects2 = ax.bar(ind + width, logistic_means, width, color='b', yerr=logistic_sde
vs)
```

```
    ax.set_ylabel('Accuracy')
    ax.set_title('Accuracy by Preprocessing and Classifier')
    ax.set_xticks(ind + width / 2)
    ax.set_xticklabels(('No Preprocessing', 'Standardization'))
    ax.legend((rects1[0], rects2[0]), ('Perceptron', 'Logistic Regression'))
    ax.set_ylim([0, 1.4])
    ax.axhline(y=dummy_mean, color='k')
    ax.axhline(y=dummy_mean-dummy_sdev, color='k', linestyle='--')
    ax.axhline(y=dummy_mean+dummy_sdev, color='k', linestyle='--')

    def autolabel(rects):
        """
        Attach a text label above each bar displaying its height
        """

        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                    '%03.3f' % height, ha='center', va='bottom')

    autolabel(rects1)
    autolabel(rects2)

    plt.show()

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