02/12/18 10:08:56
/Users/caiglencross/Documents/MachineLearning/ps2/ps4/source/phoneme.py

```python
"""
Author      : Cai Glencross & Katie Li
Class       : HMC CS 158
Date        : 2018 Feb 5
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 ========== ###
    # part b: run multiple trials of CV
    for n in range(n_trials):
        scores[n,:] = cv_performance_one_trial(clf, train_data, kfs[n])

    ### ========== 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 ========== ###
    i = 0;
    for train_index, test_index in kf.split(train_data.X):
        temp_train_data = train_data.X[train_index]
        temp_test_data = train_data.X[test_index]
        y_train = train_data.y[train_index]
        y_test = train_data.y[test_index]

        clf.fit(temp_train_data, y_train)
        y_pred_test = clf.predict(temp_test_data)
        test_accuracy = metrics.accuracy_score(y_pred_test, y_test, normalize = True)

        scores[i] = test_accuracy
        i += 1
    ### ========== TODO : END ========== ###

    return scores


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

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

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

    ### ========== TODO : START ========== ###
    # part a: is data linearly separable?
    clf = Perceptron()
    clf.fit(train_data.X, train_data.y)
    print "coefs = %s, iteration = %s" % (clf.coef_, clf.n_iter_)
    ### ========== TODO : END ========== ###

    ### ========== TODO : START ========== ###
    # part c-d: compare classifiers
    # make sure to use same folds across all runs
    N_SPLITS = 10
    N_TRIALS  = 10

    # Order: DummyClassifier, Perceptron, LogisticRegression
    # generate the kfs
    kfs = []
    for n in range(N_TRIALS):
        kf = model_selection.KFold(n_splits=N_SPLITS, shuffle=True)
        kfs.append(kf)

    descriptive_stats = {'mean':[], 'stdev':[]}
    dummyclassifier = DummyClassifier()
    perceptron  = Perceptron()
    logisticregression = LogisticRegression()
    clfs = [dummyclassifier, perceptron, logisticregression]

    score_array = []
    for clf in clfs:
        scores  = cv_performance(clf, train_data, kfs)
        mean_result = np.mean(scores)
        stdev_result = np.std(scores)
        descriptive_stats['mean'].append(mean_result)
        descriptive_stats['stdev'].append(stdev_result)
        score_array.append(scores)

    # print out descriptive_stats
    print "Descriptive stats in the order: Dummy, Perceptron, Logistic"
    print descriptive_stats

    # Do the t-test
    compare_t_tests = [[0,1],[0,2],[1,2]]

    pvalues = []
    for combo in compare_t_tests:
        result = stats.ttest_rel(score_array[combo[0]].flatten(), score_array[combo[1]].flatten())
        pvalues.append(result[1])


    print "Dummy vs Perceptron p = ", pvalues[0]
    print "Dummy vs Logistic p = ", pvalues[1]
    print "Perceptron vs Logistic p = ", pvalues[2]


    #part d: Standardization
    scaler = preprocessing.StandardScaler().fit(train_data.X)
    x_train_scaled = scaler.transform(train_data.X)
    train_data_scaled = Data(X = x_train_scaled, y= train_data.y)

    descriptive_stats_scaled = {'mean':[], 'stdev':[]}
    score_array_scaled = []
    for clf in clfs:
        scores  = cv_performance(clf, train_data_scaled, kfs)
        mean_result = np.mean(scores)
        stdev_result = np.std(scores)
        descriptive_stats_scaled['mean'].append(mean_result)
        descriptive_stats_scaled['stdev'].append(stdev_result)
        score_array_scaled.append(scores)

    pvalues_scaled = []
    for combo in compare_t_tests:
        result = stats.ttest_rel(score_array_scaled[combo[0]].flatten(), score_array_scaled[combo[1]].flatten())
        pvalues_scaled.append(result[1])

    print "Scaled Dummy vs Perceptron p = ", pvalues_scaled[0]
    print "Scaled Dummy vs Logistic p = ", pvalues_scaled[1]
    print "Scaled Perceptron vs Logistic p = ", pvalues_scaled[2]

    #t-tests for the standardized vs. non-standardized
```

```python

    #Dummy Variable Row
    pval_d = []
    for i in range(3):
        result = stats.ttest_rel(score_array_scaled[0].flatten(), score_array[i].flatten())
        pval_d.append(result[1])

    print "Dummy Row vs Elements in Standardized"
    print "D vs Standard D", pval_d[0]
    print "D vs Standard P", pval_d[1]
    print "D vs Standard L", pval_d[2]


    #Perceptron Variable Row
    pval_p = []
    for i in range(3):
        result = stats.ttest_rel(score_array_scaled[1].flatten(), score_array[i].flatten())
        pval_p.append(result[1])

    print "Perceptron Row vs Elements in Standardized"
    print "P vs Standard D", pval_p[0]
    print "P vs Standard P", pval_p[1]
    print "P vs Standard L", pval_p[2]

    #Logistic Regression Variable Row
    pval_r = []
    for i in range(3):
        result = stats.ttest_rel(score_array_scaled[2].flatten(), score_array[i].flatten())
        pval_r.append(result[1])

    print "Logistic Row vs Elements in Standardized"
    print "L vs Standard D", pval_r[0]
    print "L vs Standard P", pval_r[1]
    print "L vs Standard L", pval_r[2]

    # print out descriptive_stats
    print "Descriptive scaled stats in the order: Dummy, Perceptron, Logistic"
    print descriptive_stats_scaled


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

    ### ========== TODO : START ========== ###
    # part e: plot

    # Indicies of descriptive stats: 0 - Dummy, 1 - Perceptron, 2 -
```

```
Logistic

    N = 2
    ind = np.arange(N)  # the x locations for the groups
    width = 0.35       # the width of the bars
    fig, ax = plt.subplots()


    dum_mean = descriptive_stats['mean'][0]
    dum_stdev = descriptive_stats['stdev'][0]

    percep_means = (descriptive_stats['mean'][1],
                    descriptive_stats_scaled['mean'][1])
    percep_stdev = (descriptive_stats['stdev'][1],
                    descriptive_stats_scaled['stdev'][1])
    rects1 = ax.bar(ind + width, percep_means, width, color='r', yerr=percep_stdev)

    log_reg_means = (descriptive_stats['mean'][2],
                    descriptive_stats_scaled['mean'][2])
    log_reg_stdev = (descriptive_stats['stdev'][2],
            descriptive_stats_scaled['stdev'][2])
    rects2 = ax.bar(ind, log_reg_means, width, color='b', yerr=log_reg_stdev)



    # add some text for labels, title and axes ticks
    ax.set_ylabel('Accuracy')
    ax.set_title('Accuracies 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'), loc='lower right')


    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,
                '%.2f' % height,
                ha='center', va='bottom')


    autolabel(rects1)
    autolabel(rects2)
```

```python

    plt.axhline(y=dum_mean, color ='k', linestyle='solid')
    plt.axhline(y=(dum_mean + dum_stdev),color='k', linestyle='dashed')
    plt.axhline(y=(dum_mean - dum_stdev),color='k', linestyle='dashed')

    plt.show()

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

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