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

```
  1 """
  2 Author      : Cai Glencross & Katie Li
  3 Class       : HMC CS 158
  4 Date        : 2018 Feb 5
  5 Description : Perceptron vs Logistic Regression on a Phoneme Dataset
  6 """
  7
  8 # utilities
  9 from util import *
 10
 11 # scipy libraries
 12 from scipy import stats
 13
 14 # scikit-learn libraries
 15 from sklearn import preprocessing
 16 from sklearn import metrics
 17 from sklearn import model_selection
 18 from sklearn.dummy import DummyClassifier
 19 from sklearn.linear_model import Perceptron, LogisticRegression
 20
 21 ##########################################################################
    ##
 22 # functions
 23 ##########################################################################
    ##
 24
 25 def cv_performance(clf, train_data, kfs) :
 26     """
 27     Determine classifier performance across multiple trials using
    cross-validation
 28
 29     Parameters
 30     --------------------
 31         clf        -- classifier
 32         train_data -- Data, training data
 33         kfs        -- array of size n_trials
 34                       each element is one fold from
    model_selection.KFold
 35
 36     Returns
 37     --------------------
 38         scores     -- numpy array of shape (n_trials, n_fold)
 39                       each element is the (accuracy) score of one
    fold in one trial
 40     """
 41
```

```
42      n_trials = len(kfs)
43      n_folds = kfs[0].n_splits
44      scores = np.zeros((n_trials, n_folds))
45
46      ### ========== TODO : START ========== ###
47      # part b: run multiple trials of CV
48      for n in range(n_trials):
49          scores[n,:] = cv_performance_one_trial(clf, train_data,
    kfs[n])
50
51      ### ========== TODO : END ========== ###
52
53      return scores
54
55
56  def cv_performance_one_trial(clf, train_data, kf) :
57      """
58      Compute classifier performance across multiple folds using
    cross-validation
59
60      Parameters
61      --------------------
62          clf         -- classifier
63          train_data -- Data, training data
64          kf          -- model_selection.KFold
65
66      Returns
67      --------------------
68          scores      -- numpy array of shape (n_fold, )
69                         each element is the (accuracy) score of one
    fold
70      """
71
72      scores = np.zeros(kf.n_splits)
73
74      ### ========== TODO : START ========== ###
75      i = 0;
76      for train_index, test_index in kf.split(train_data.X):
77          temp_train_data = train_data.X[train_index]
78          temp_test_data = train_data.X[test_index]
79          y_train = train_data.y[train_index]
80          y_test = train_data.y[test_index]
81
82          clf.fit(temp_train_data, y_train)
83          y_pred_test = clf.predict(temp_test_data)
84          test_accuracy = metrics.accuracy_score(y_pred_test, y_test,
    normalize = True)
85
86          scores[i] = test_accuracy
87          i += 1
```

```
88      ### ========== TODO : END ========== ###
89
90      return scores
91
92
93  ##########################################################################
    ##
94  # main
95  ##########################################################################
    ##
96
97  def main() :
98      np.random.seed(1234)
99
100     #========================================
101     # load data
102     train_data = load_data("phoneme_train.csv")
103
104     ### ========== TODO : START ========== ###
105     # part a: is data linearly separable?
106     clf = Perceptron()
107     clf.fit(train_data.X, train_data.y)
108     print "coefs = %s, iteration = %s" % (clf.coef_, clf.n_iter_)
109     ### ========== TODO : END ========== ###
110
111     ### ========== TODO : START ========== ###
112     # part c-d: compare classifiers
113     # make sure to use same folds across all runs
114     N_SPLITS = 10
115     N_TRIALS  = 10
116
117     # Order: DummyClassifier, Perceptron, LogisticRegression
118     # generate the kfs
119     kfs = []
120     for n in range(N_TRIALS):
121         kf = model_selection.KFold(n_splits=N_SPLITS, shuffle=True)
122         kfs.append(kf)
123
124     descriptive_stats = {'mean':[], 'stdev':[]}
125     dummyclassifier = DummyClassifier()
126     perceptron  = Perceptron()
127     logisticregression = LogisticRegression()
128     clfs = [dummyclassifier, perceptron, logisticregression]
129
130     score_array = []
131     for clf in clfs:
132         scores  = cv_performance(clf, train_data, kfs)
133         mean_result = np.mean(scores)
134         stdev_result = np.std(scores)
135         descriptive_stats['mean'].append(mean_result)
```

```
136         descriptive_stats['stdev'].append(stdev_result)
137         score_array.append(scores)
138 
139     # print out descriptive_stats
140     print "Descriptive stats in the order: Dummy, Perceptron,
    Logistic"
141     print descriptive_stats
142 
143     # Do the t-test
144     compare_t_tests = [[0,1],[0,2],[1,2]]
145 
146     pvalues = []
147     for combo in compare_t_tests:
148         result = stats.ttest_rel(score_array[combo[0]].flatten(),
    score_array[combo[1]].flatten())
149         pvalues.append(result[1])
150 
151 
152     print "Dummy vs Perceptron p = ", pvalues[0]
153     print "Dummy vs Logistic p = ", pvalues[1]
154     print "Perceptron vs Logistic p = ", pvalues[2]
155 
156 
157     #part d: Standardization
158     scaler = preprocessing.StandardScaler().fit(train_data.X)
159     x_train_scaled = scaler.transform(train_data.X)
160     train_data_scaled = Data(X = x_train_scaled, y= train_data.y)
161 
162     descriptive_stats_scaled = {'mean':[], 'stdev':[]}
163     score_array_scaled = []
164     for clf in clfs:
165         scores  = cv_performance(clf, train_data_scaled, kfs)
166         mean_result = np.mean(scores)
167         stdev_result = np.std(scores)
168         descriptive_stats_scaled['mean'].append(mean_result)
169         descriptive_stats_scaled['stdev'].append(stdev_result)
170         score_array_scaled.append(scores)
171 
172     pvalues_scaled = []
173     for combo in compare_t_tests:
174         result =
    stats.ttest_rel(score_array_scaled[combo[0]].flatten(),
    score_array_scaled[combo[1]].flatten())
175         pvalues_scaled.append(result[1])
176 
177     print "Scaled Dummy vs Perceptron p = ", pvalues_scaled[0]
178     print "Scaled Dummy vs Logistic p = ", pvalues_scaled[1]
179     print "Scaled Perceptron vs Logistic p = ", pvalues_scaled[2]
180 
181     #t-tests for the standardized vs. non-standardized
```

```
182
183     #Dummy Variable Row
184     pval_d = []
185     for i in range(3):
186         result = stats.ttest_rel(score_array_scaled[0].flatten(),
    score_array[i].flatten())
187         pval_d.append(result[1])
188
189     print "Dummy Row vs Elements in Standardized"
190     print "D vs Standard D", pval_d[0]
191     print "D vs Standard P", pval_d[1]
192     print "D vs Standard L", pval_d[2]
193
194
195     #Perceptron Variable Row
196     pval_p = []
197     for i in range(3):
198         result = stats.ttest_rel(score_array_scaled[1].flatten(),
    score_array[i].flatten())
199         pval_p.append(result[1])
200
201     print "Perceptron Row vs Elements in Standardized"
202     print "P vs Standard D", pval_p[0]
203     print "P vs Standard P", pval_p[1]
204     print "P vs Standard L", pval_p[2]
205
206     #Logistic Regression Variable Row
207     pval_r = []
208     for i in range(3):
209         result = stats.ttest_rel(score_array_scaled[2].flatten(),
    score_array[i].flatten())
210         pval_r.append(result[1])
211
212     print "Logistic Row vs Elements in Standardized"
213     print "L vs Standard D", pval_r[0]
214     print "L vs Standard P", pval_r[1]
215     print "L vs Standard L", pval_r[2]
216
217     # print out descriptive_stats
218     print "Descriptive scaled stats in the order: Dummy, Perceptron,
    Logistic"
219     print descriptive_stats_scaled
220
221
222     ### ========== TODO : END ========== ###
223
224     ### ========== TODO : START ========== ###
225     # part e: plot
226
227     # Indicies of descriptive stats: 0 - Dummy, 1 - Perceptron, 2 -
```

```
    Logistic
228
229     N = 2
230     ind = np.arange(N)  # the x locations for the groups
231     width = 0.35       # the width of the bars
232     fig, ax = plt.subplots()
233
234
235     dum_mean = descriptive_stats['mean'][0]
236     dum_stdev = descriptive_stats['stdev'][0]
237
238     percep_means = (descriptive_stats['mean'][1],
239                     descriptive_stats_scaled['mean'][1])
240     percep_stdev = (descriptive_stats['stdev'][1],
241                     descriptive_stats_scaled['stdev'][1])
242     rects1 = ax.bar(ind + width, percep_means, width, color='r',
    yerr=percep_stdev)
243
244     log_reg_means = (descriptive_stats['mean'][2],
245                     descriptive_stats_scaled['mean'][2])
246     log_reg_stdev = (descriptive_stats['stdev'][2],
247             descriptive_stats_scaled['stdev'][2])
248     rects2 = ax.bar(ind, log_reg_means, width, color='b',
    yerr=log_reg_stdev)
249
250
251
252     # add some text for labels, title and axes ticks
253     ax.set_ylabel('Accuracy')
254     ax.set_title('Accuracies by Preprocessing and Classifier')
255     ax.set_xticks(ind + width / 2)
256     ax.set_xticklabels(('No Preprocessing', 'Standardization'))
257
258     ax.legend((rects1[0], rects2[0]), ('Perceptron', 'Logistic
    Regression'), loc='lower right')
259
260
261     def autolabel(rects):
262         """
263         Attach a text label above each bar displaying its height
264         """
265         for rect in rects:
266             height = rect.get_height()
267             ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
268                 '%.2f' % height,
269                 ha='center', va='bottom')
270
271
272     autolabel(rects1)
273     autolabel(rects2)
```

```
274
275     plt.axhline(y=dum_mean, color ='k', linestyle='solid')
276     plt.axhline(y=(dum_mean + dum_stdev),color='k',
    linestyle='dashed')
277     plt.axhline(y=(dum_mean - dum_stdev),color='k',
    linestyle='dashed')
278
279     plt.show()
280
281     ### ========== TODO : END ========== ###
282
283 if __name__ == "__main__" :
284     main()
```