01/30/18 10:11:36 /Users/caiglencross/Documents/MachineLearning/ps2/source/titanic.py

```
  1 """
  2 Author      : Cai Glencross and Katie Li
  3 Class       : HMC CS 158
  4 Date        : 2018 Jan 30
  5 Description : Titanic
  6 """
  7
  8 # Use only the provided packages!
  9 import math
 10 import csv
 11 from util import *
 12
 13 from sklearn.tree import DecisionTreeClassifier
 14 from sklearn.model_selection import train_test_split
 15 from sklearn import metrics
 16
 17 ##################################################################
 18 # classes
 19 ##################################################################
 20
 21 class Classifier(object) :
 22     """
 23     Classifier interface.
 24     """
 25
 26     def fit(self, X, y):
 27         raise NotImplementedError()
 28
 29     def predict(self, X):
 30         raise NotImplementedError()
 31
 32
 33 class MajorityVoteClassifier(Classifier) :
 34
 35     def __init__(self) :
 36         """
 37         A classifier that always predicts the majority class.
 38
 39         Attributes
 40         --------------------
 41             prediction_ -- majority class
 42         """
 43         self.prediction_ = None
 44
 45     def fit(self, X, y) :
 46         """
 47         Build a majority vote classifier from the training set (X, y).
 48
 49         Parameters
 50         --------------------
 51             X    -- numpy array of shape (n,d), samples
 52             y    -- numpy array of shape (n,), target classes
 53
 54         Returns
 55         --------------------
 56             self -- an instance of self
 57         """
```

```
58          vals, counts = np.unique(y, return_counts=True)
59          majority_val, majority_count = max(zip(vals, counts), key=lambda (val,
    count): count)
60          self.prediction_ = majority_val
61          return self
62
63      def predict(self, X) :
64          """
65          Predict class values.
66
67          Parameters
68          --------------------
69              X    -- numpy array of shape (n,d), samples
70
71          Returns
72          --------------------
73              y    -- numpy array of shape (n,), predicted classes
74          """
75          if self.prediction_ is None :
76              raise Exception("Classifier not initialized. Perform a fit first.")
77
78          n,d = X.shape
79          y = [self.prediction_] * n
80          return y
81
82
83  class RandomClassifier(Classifier) :
84
85      def __init__(self) :
86          """
87          A classifier that predicts according to the distribution of the classes.
88
89          Attributes
90          --------------------
91              probabilities_ -- class distribution dict (key = class, val =
    probability of class)
92          """
93          self.probabilities_ = None
94
95      def fit(self, X, y) :
96          """
97          Build a random classifier from the training set (X, y).
98
99          Parameters
100         --------------------
101             X    -- numpy array of shape (n,d), samples
102             y    -- numpy array of shape (n,), target classes
103
104         Returns
105         --------------------
106             self -- an instance of self
107         """
108
109         ### ========== TODO : START ========== ###
110
111         # part c: set self.probabilities_ according to the training set
112
113         vals, counts = np.unique(y, return_counts=True)
114         majority_val, majority_count = max(zip(vals, counts), key=lambda (val,
    count): count)
```

```
115         minority_val, minority_count = min(zip(vals, counts), key=lambda (val,
    count): count)
116 
117         # find the highest probability
118         self.probabilities_ = {majority_val: float(majority_count)/sum(counts),
119                                minority_val: float(minority_count)/sum(counts)}
120 
121         ### ========== TODO : END ========== ###
122 
123         return self
124 
125     def predict(self, X, seed=1234) :
126         """
127         Predict class values.
128 
129         Parameters
130         --------------------
131             X    -- numpy array of shape (n,d), samples
132             seed -- integer, random seed
133 
134         Returns
135         --------------------
136             y    -- numpy array of shape (n,), predicted classes
137         """
138         if self.probabilities_ is None :
139             raise Exception("Classifier not initialized. Perform a fit first.")
140         np.random.seed(seed)
141 
142         ### ========== TODO : START ========== ###
143         # part c: predict the class for each test example
144         # hint: use np.random.choice (be careful of the parameters)
145 
146         class_distr = self.probabilities_
147         n, d = X.shape
148 
149         # Create a random distribution based on given probabilities of survival
150         random_ys = np.random.choice(class_distr.keys(), n, p =
    class_distr.values())
151         y = random_ys
152         ### ========== TODO : END ========== ###
153 
154         return y
155 
156 
157 ##############################################################################
158 # functions
159 ##############################################################################
160 
161 def error(clf, X, y, ntrials=100, test_size=0.2) :
162     """
163     Computes the classifier error over a random split of the data,
164     averaged over ntrials runs.
165 
166     Parameters
167     --------------------
168         clf         -- classifier
169         X           -- numpy array of shape (n,d), features values
170         y           -- numpy array of shape (n,), target classes
171         ntrials     -- integer, number of trials
172         test_size   -- float (between 0.0 and 1.0) or int,
```

```
173                         if float, the proportion of the dataset to include in the
    test split
174                         if int, the absolute number of test samples
175
176     Returns
177     --------------------
178         train_error -- float, training error
179         test_error  -- float, test error
180     """
181
182     ### ========== TODO : START ========== ###
183     # part b: compute cross-validation error over ntrials
184     # hint: use train_test_split (be careful of the parameters)
185     SEED = 1234
186     np.random.seed(SEED)
187
188     train_error = 0
189     test_error = 0
190
191     test_error_total = 0
192     train_error_total = 0
193
194     n, d = X.shape
195
196
197
198     for t in range (0, ntrials):
199         # get the split of data
200         X_train = np.empty((0,d)); X_test = np.empty((0,d))
201         y_train = []; y_test = []
202
203         for i in range(0, n):
204             if (np.random.random_sample() < (1.0 - test_size)):
205                 X_train = np.vstack((X_train, X[i,:]))
206                 y_train.append(y[i])
207             else:
208                 X_test = np.vstack((X_test,X[i,:]))
209                 y_test.append(y[i])
210
211         # print "X_train shape is", X_train.shape
212         # print "shape of first element of X_train is:", X[0,:].shape
213         # print "shape of second elt:", X[1,:].shape
214         # print "shape of stack is:", np.vstack((X[0,:],X[1,:])).shape
215         # print "y train length is:", len(y)
216         clf.fit(X_train,y_train)
217         y_pred_test = clf.predict(X_test)
218         y_pred_train = clf.predict(X_train)
219         test_error_total += 1 - metrics.accuracy_score(y_pred_test, y_test,
    normalize=True)
220         train_error_total += 1 - metrics.accuracy_score(y_pred_train, y_train,
    normalize=True)
221
222     train_error = train_error_total/ntrials
223     test_error = test_error_total/ntrials
224     ### ========== TODO : END ========== ###
225
226     return train_error, test_error
227
228
229 def write_predictions(y_pred, filename, yname=None) :
```

```
230     """Write out predictions to csv file."""
231     out = open(filename, 'wb')
232     f = csv.writer(out)
233     if yname :
234         f.writerow([yname])
235     f.writerows(zip(y_pred))
236     out.close()
237 
238 
239 ##############################################################################
240 # main
241 ##############################################################################
242 
243 def main():
244     # load Titanic dataset
245     titanic = load_data("titanic_train.csv", header=1, predict_col=0)
246     X = titanic.X; Xnames = titanic.Xnames
247     y = titanic.y; yname = titanic.yname
248     n,d = X.shape  # n = number of examples, d =  number of features
249 
250 
251 
252     #========================================
253     # train Majority Vote classifier on data
254     print 'Classifying using Majority Vote...'
255     clf = MajorityVoteClassifier() # create MajorityVote classifier, which
    includes all model parameters
256     clf.fit(X, y)                  # fit training data using the classifier
257     y_pred = clf.predict(X)        # take the classifier and run it on the
    training data
258     train_error = 1 - metrics.accuracy_score(y, y_pred, normalize=True)
259     print '\t-- training error: %.3f' % train_error
260 
261 
262 
263     ### ========== TODO : START ========== ###
264     # part a: evaluate training error of Decision Tree classifier
265     print 'Classifying using Decision Tree...'
266     clf = DecisionTreeClassifier(criterion="entropy")
267     clf.fit(X, y)
268     y_pred = clf.predict(X)
269     train_error = 1 - metrics.accuracy_score(y, y_pred, normalize=True)
270     print '\t-- training error: %.3f' % train_error
271 
272     ### ========== TODO : END ========== ###
273 
274 
275 
276     # note: uncomment out the following lines to output the Decision Tree graph
277     """
278     # save the classifier -- requires GraphViz and pydot
279     import StringIO, pydot
280     from sklearn import tree
281     dot_data = StringIO.StringIO()
282     tree.export_graphviz(clf, out_file=dot_data,
283                          feature_names=Xnames,
284                          class_names=["Died", "Survived"])
285     graph = pydot.graph_from_dot_data(dot_data.getvalue())
286     graph.write_pdf("dtree.pdf")
287     """
```

```
288
289
290
291     ### ========== TODO : START ========== ###
292     # part b: use cross-validation to compute average training and test error of
    classifiers
293     print 'Investigating various classifiers...'
294
295     print 'MajorityVoteClassifier Error'
296     train_error, test_error = error(MajorityVoteClassifier() , X, y,
    ntrials=100, test_size=0.2)
297     print '\t-- training error: %.3f' % train_error
298     print '\t-- test error: %.3f' % test_error
299
300     print 'RandomClassifier Error'
301     train_error, test_error = error(RandomClassifier() , X, y, ntrials=100,
    test_size=0.2)
302     print '\t-- training error: %.3f' % train_error
303     print '\t-- test error: %.3f' % test_error
304
305     print 'DecisionTreeClassifier Error'
306     train_error, test_error = error(DecisionTreeClassifier(criterion="entropy"),
    X, y, ntrials=100, test_size=0.2)
307     print '\t-- training error: %.3f' % train_error
308     print '\t-- test error: %.3f' % test_error
309
310
311
312     ### ========== TODO : END ========== ###
313
314
315
316     ### ========== TODO : START ========== ###
317     # part c: investigate decision tree classifier with various depths
318     print 'Investigating depths...'
319     clfs = [MajorityVoteClassifier(), RandomClassifier()]
320     label_clfs = ["MVC", "RC"]
321     for clf_index in range(0,len(clfs)):
322         train_error_arr = []
323         test_error_arr =[]
324         depths = range(1,21)
325         for depth in depths:
326             clf = clfs[clf_index]
327             train_error, test_error = error(clf,X,y)
328             train_error_arr.append(train_error)
329             test_error_arr.append(test_error)
330
331         plt.plot(depths, train_error_arr, label = label_clfs[clf_index] + "
    train error")
332         plt.plot(depths, test_error_arr, label = label_clfs[clf_index] + " test
    error")
333
334     train_error_arr = []
335     test_error_arr =[]
336     # only do the DecisionTreeClassifier One
337     for depth in depths:
338         clf = DecisionTreeClassifier(criterion="entropy", max_depth=depth)
339         train_error, test_error = error(clf,X,y)
340         train_error_arr.append(train_error)
341         test_error_arr.append(test_error)
```

```
342
343     plt.plot(depths, train_error_arr, label = "DT train error")
344     plt.plot(depths, test_error_arr, label = "DT test error")
345     plt.legend()
346     plt.xlabel("Max Depth")
347     plt.ylabel("Average Error")
348     plt.show()
349     ### ========== TODO : END ========== ###
350
351
352
353
354     ### ========== TODO : START ========== ###
355     # part d: investigate decision tree classifier with various training set
    sizes
356     print 'Investigating training set sizes...'
357
358     clfs = [MajorityVoteClassifier(), RandomClassifier(),
    DecisionTreeClassifier(criterion="entropy", max_depth=6)]
359     label_clfs = ["MVC", "RC", "DT"]
360     split_sizes =  map (lambda u: u*.05,  range(1,20))
361     training_splits = map (lambda u: 1 - u,  split_sizes)
362
363     for clf_index in range(0, len(clfs)):
364         train_error_arr = []
365         test_error_arr =[]
366         for split in split_sizes:
367             clf = clfs[clf_index]
368             train_error, test_error = error(clf,X,y,test_size=split)
369             train_error_arr.append(train_error)
370             test_error_arr.append(test_error)
371
372         plt.plot(training_splits, train_error_arr, label = label_clfs[clf_index]
    + " train error")
373         plt.plot(training_splits, test_error_arr, label = label_clfs[clf_index]
    +" test error")
374
375     plt.xlabel("Training Data Size")
376     plt.ylabel("Average Error")
377     plt.legend()
378     plt.show()
379
380
381     ### ========== TODO : END ========== ###
382
383
384
385     ### ========== TODO : START ========== ###
386     # Contest
387     # uncomment write_predictions and change the filename
388
389     # evaluate on test data
390     titanic_test = load_data("titanic_test.csv", header=1, predict_col=None)
391     X_test = titanic_test.X
392     y_pred = clf.predict(X_test)   # take the trained classifier and run it on
    the test data
393     #write_predictions(y_pred, "../data/yjw_titanic.csv", titanic.yname)
394
395     ### ========== TODO : END ========== ###
396     print 'Done'
```

```
397
398 if __name__ == "__main__":
399     main()
```