

03/04/18 03:45:11

/Users/caiglencross/Documents/MachineLearning/ps2/ps6/source/twitter.py

```

1  """
2  Author      : Cai Glencross
3  Class       : HMC CS 158
4  Date        : 2018 Feb 14
5  Description : Twitter
6  """
7
8  """
9  Author: Cai Glencross, Katie Li
10 """
11
12 from string import punctuation
13
14 # numpy libraries
15 import numpy as np
16
17 # matplotlib libraries
18 import matplotlib
19 matplotlib.use('TkAgg')
20 import matplotlib.pyplot as plt
21
22 # scikit-learn libraries
23 from sklearn.dummy import DummyClassifier
24 from sklearn.svm import SVC
25 from sklearn.model_selection import StratifiedKFold
26 from sklearn import metrics
27 from sklearn.utils import shuffle
28
29 #####
30 ##
31 # functions -- input/output
32 #####
33 ##
34
35 def read_vector_file(fname) :
36     """
37     Reads and returns a vector from a file.
38
39     Parameters
40     -----
41     fname -- string, filename
42
43     Returns
44     -----
45     labels -- numpy array of shape (n,)
46             n is the number of non-blank lines in the text

```

```

file
45     """
46     return np.genfromtxt(fname)
47
48
49 def write_label_answer(vec, outfile) :
50     """
51     Writes your label vector to the given file.
52
53     Parameters
54     -----
55     vec      -- numpy array of shape (n,) or (n,1), predicted
56 scores
57     outfile -- string, output filename
58     """
59     # for this project, you should predict 70 labels
60     if(vec.shape[0] != 70):
61         print("Error - output vector should have 70 rows.")
62         print("Aborting write.")
63         return
64
65     np.savetxt(outfile, vec)
66
67
68 #####
69 ##
69 # functions -- feature extraction
70 #####
71 ##
72 def extract_words(input_string) :
73     """
74     Processes the input_string, separating it into "words" based on
75 the presence
76 of spaces, and separating punctuation marks into their own
77 words.
78
79     Parameters
80     -----
81     input_string -- string of characters
82
83     Returns
84     -----
85     words      -- list of lowercase "words"
86     """
87
88     for c in punctuation :
89         input_string = input_string.replace(c, ' ' + c + ' ')
90     return input_string.lower().split()

```

```

89
90
91 def extract_dictionary(infile) :
92     """
93     Given a filename, reads the text file and builds a dictionary of
unique
94     words/punctuations.
95
96     Parameters
97     -----
98     infile      -- string, filename
99
100    Returns
101    -----
102    word_list -- dictionary, (key, value) pairs are (word,
index)
103    """
104
105    word_list = {}
106    with open(infile, 'rU') as fid :
107        ### ===== TODO : START ===== ###
108        # part 1a: process each line to populate word_list
109        index = 0
110        for input_string in fid:
111            words = extract_words(input_string)
112
113            for word in words:
114                if (not (word in word_list)):
115                    word_list[word] = index
116                    index = index + 1
117        ### ===== TODO : END ===== ###
118
119    return word_list
120
121
122 def extract_feature_vectors(infile, word_list) :
123     """
124     Produces a bag-of-words representation of a text file specified
by the
125     filename infile based on the dictionary word_list.
126
127     Parameters
128     -----
129     infile      -- string, filename
130     word_list    -- dictionary, (key, value) pairs are (word,
index)
131
132     Returns
133     -----
134     feature_matrix -- numpy array of shape (n,d)

```

```

135         boolean (0,1) array indicating word
presence in a string
136         n is the number of non-blank lines in
the text file
137         d is the number of unique words in the
text file
138         """
139
140     num_lines = sum(1 for line in open(infile, 'rU'))
141     num_words = len(word_list)
142     feature_matrix = np.zeros((num_lines, num_words))
143
144     with open(infile, 'rU') as fid :
145         ### ===== TODO : START ===== ###
146         # part 1b: process each line to populate feature_matrix
147         n = 0 # set index of line number
148         for input_string in fid:
149             words = extract_words(input_string)
150             for word in words:
151                 index = word_list[word]
152                 feature_matrix[n, index] = 1
153             n = n + 1
154         ### ===== TODO : END ===== ###
155
156     return feature_matrix
157
158
159 def test_extract_dictionary(dictionary) :
160     err = "extract_dictionary implementation incorrect"
161
162     assert len(dictionary) == 1811, err
163
164     exp = [('2012', 0),
165            ('carol', 10),
166            ('ve', 20),
167            ('scary', 30),
168            ('vacation', 40),
169            ('just', 50),
170            ('excited', 60),
171            ('no', 70),
172            ('cinema', 80),
173            ('frm', 90)]
174     act = [sorted(dictionary.items(), key=lambda it: it[1])[i] for i
in range(0,100,10)]
175     assert exp == act, err
176
177
178 def test_extract_feature_vectors(X) :
179     err = "extract_features_vectors implementation incorrect"
180

```

```

181     assert X.shape == (630, 1811), err
182
183     exp = np.array([[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
184 1.],
185                     [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
186 1.],
187                     [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
188 1.],
189                     [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,
190 1.],
191                     [ 0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,
192 1.],
193                     [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,
194 1.],
195                     [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
196 1.],
197                     [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
198 1.],
199                     [ 0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,
200 1.],
201                     [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
202 1.]])
203     act = X[:10,:10]
204     assert (exp == act).all(), err
205
206 #####
207 ##
208 # functions -- evaluation
209 #####
210 ##
211
212 def performance(y_true, y_pred, metric="accuracy") :
213     """
214     Calculates the performance metric based on the agreement between
215     the
216     true labels and the predicted labels.
217
218     Parameters
219     -----
220         y_true -- numpy array of shape (n,), known labels
221         y_pred -- numpy array of shape (n,), (continuous-valued)
222         predictions
223         metric -- string, option used to select the performance
224         measure
225         options: 'accuracy', 'f1_score', 'auroc',
226         'precision',
227         'sensitivity', 'specificity'
228
229     Returns

```

```

215 -----
216         score -- float, performance score
217     """
218     # map continuous-valued predictions to binary labels
219     y_label = np.sign(y_pred)
220     y_label[y_label==0] = 1 # map points of hyperplane to +1
221
222     ### ===== TODO : START ===== ###
223     # part 2a: compute classifier performance
224
225     tn, fp, fn, tp = metrics.confusion_matrix(y_true,
y_label).ravel()
226
227     if (metric == "accuracy"):
228         accuracy = metrics.accuracy_score(y_true, y_label, normalize
= True)
229         selected_metric = accuracy
230     elif (metric == "f1_score"):
231         f1score = metrics.f1_score(y_true, y_label)
232         selected_metric = f1score
233     elif (metric == "precision"):
234         precision = metrics.precision_score(y_true, y_label)
235         selected_metric = precision
236     elif (metric == "auroc"):
237         auroc = metrics.roc_auc_score(y_true, y_pred)
238         selected_metric = auroc
239     elif (metric == "sensitivity"):
240         sensitivity = float(tp)/(tp + fn)
241         selected_metric = sensitivity
242     elif (metric == "specificity"):
243         specificity = float(tn) / (tn + fp)
244         selected_metric = specificity
245
246     return selected_metric
247     ### ===== TODO : END ===== ###
248
249 def test_performance() :
250     # np.random.seed(1234)
251     # y_true = 2 * np.random.randint(0,2,10) - 1
252     # np.random.seed(2345)
253     # y_pred = (10 + 10) * np.random.random(10) - 10
254
255     y_true = [ 1, 1, -1, 1, -1, -1, -1, 1, 1, 1]
256     #y_pred = [ 1, -1, 1, -1, 1, 1, -1, -1, 1, -1]
257     # confusion matrix
258     #           pred pos      neg
259     # true pos      tp (2)  fn (4)
260     #      neg      fp (3)  tn (1)
261     y_pred = [ 3.21288618, -1.72798696, 3.36205116, -5.40113156,
6.15356672,

```

```

262         2.73636929, -6.55612296, -4.79228264, 8.30639981,
-0.74368981]
263     metrics = ["accuracy", "f1_score", "auroc", "precision",
"sensitivity", "specificity"]
264     scores = [ 3/10., 4/11., 5/12.,
2/5., 2/6., 1/4.]
265
266     import sys
267     eps = sys.float_info.epsilon
268
269     for i, metric in enumerate(metrics) :
270         assert abs(performance(y_true, y_pred, metric) - scores[i])
< eps, \
271             (metric, performance(y_true, y_pred, metric), scores[i])
272
273
274 def cv_performance(clf, X, y, kf, metric="accuracy") :
275     """
276     Splits the data, X and y, into k-folds and runs k-fold cross-
validation.
277     Trains classifier on k-1 folds and tests on the remaining fold.
278     Calculates the k-fold cross-validation performance metric for
classifier
279     by averaging the performance across folds.
280
281     Parameters
282     -----
283     clf      -- classifier (instance of SVC)
284     X        -- numpy array of shape (n,d), feature vectors
285                 n = number of examples
286                 d = number of features
287     y        -- numpy array of shape (n,), binary labels {1,-1}
288     kf       -- model_selection.KFold or
model_selection.StratifiedKFold
289     metric   -- string, option used to select performance measure
290
291     Returns
292     -----
293     score    -- float, average cross-validation performance
across k folds
294     """
295
296     scores = []
297     for train, test in kf.split(X, y) :
298         X_train, X_test, y_train, y_test = X[train], X[test],
y[train], y[test]
299         clf.fit(X_train, y_train)
300         # use SVC.decision_function to make ``continuous-valued''
predictions
301         y_pred = clf.decision_function(X_test)

```

```

302     score = performance(y_test, y_pred, metric)
303     if not np.isnan(score) :
304         scores.append(score)
305     return np.array(scores).mean()
306
307
308 def select_param_linear(X, y, kf, metric="accuracy", plot=True) :
309     """
310     Sweeps different settings for the hyperparameter of a linear-
311     kernel SVM,
312     calculating the k-fold CV performance for each setting, then
313     selecting the
314     hyperparameter that 'maximize' the average k-fold CV
315     performance.
316
317     Parameters
318     -----
319     X      -- numpy array of shape (n,d), feature vectors
320             n = number of examples
321             d = number of features
322     y      -- numpy array of shape (n,), binary labels {1,-1}
323     kf      -- model_selection.KFold or
324               model_selection.StratifiedKFold
325     metric -- string, option used to select performance measure
326     plot    -- boolean, make a plot
327
328     Returns
329     -----
330     C -- float, optimal parameter value for linear-kernel SVM
331     """
332
333     print 'Linear SVM Hyperparameter Selection based on ' +
334     str(metric) + ':'
335     C_range = 10.0 ** np.arange(-3, 3)
336
337     ### ===== TODO : START ===== ###
338     # part 2c: select optimal hyperparameter using cross-validation
339     best_c = (-1,-1) # (score, C_best)
340     scores = []
341     for c_i in C_range:
342         svm = SVC(C = c_i, kernel='linear')
343         score = cv_performance(svm, X, y, kf, metric=metric)
344         scores.append(score)
345         if (best_c[0] == -1 ) or (score > best_c[0]):
346             best_c = (score, c_i)
347
348     if plot:
349         lineplot(C_range, scores, metric)
350         plt.hold()
351

```



```

347     print(best_c[1]) # print out the optimal hyperparameter score
348     return best_c[1]
349     ### ===== TODO : END ===== ###
350
351
352 def plot_metric_2d(X, y, kf) :
353     """
354     Plots line plots of all the metrics
355
356     Parameters
357     -----
358         X      -- numpy array of shape (n,d), feature vectors
359                 n = number of examples
360                 d = number of features
361         y      -- numpy array of shape (n,), binary labels {1,-1}
362         kf     -- model_selection.KFold or
model_selection.StratifiedKFold
363
364     Action
365     -----
366     creates a line plot
367     """
368
369     C_range = 10.0 ** np.arange(-3, 3)
370     metric_list = ["accuracy", "f1_score", "auroc", "precision",
"sensitivity", "specificity"]
371
372     ### ===== TODO : START ===== ###
373     # part 2c: select optimal hyperparameter using cross-validation
374     for metric in metric_list:
375         scores = []
376         for c_i in C_range:
377             svm = SVC(C = c_i, kernel = 'linear')
378             score = cv_performance(svm, X, y, kf, metric=metric)
379             scores.append(score)
380
381             xx = range(len(scores))
382             plt.plot(xx, scores, linestyle='-', linewidth=2,
label=metric)
383             plt.xticks(xx, C_range)
384             plt.xlabel("C")
385             plt.ylabel("Scores")
386             plt.title("Classifier Performance")
387             plt.legend()
388
389     plt.show()
390
391     ### ===== TODO : END ===== ###
392
393 def select_param_rbf(X, y, kf, metric="accuracy") :

```

```

394     """
395     Sweeps different settings for the hyperparameters of an RBF-
kernel SVM,
396     calculating the k-fold CV performance for each setting, then
selecting the
397     hyperparameters that 'maximize' the average k-fold CV
performance.
398
399     Parameters
400     -----
401     X          -- numpy array of shape (n,d), feature vectors
402                  n = number of examples
403                  d = number of features
404     y          -- numpy array of shape (n,), binary labels {1,-1}
405     kf         -- model_selection.KFold or
model_selection.StratifiedKFold
406     metric     -- string, option used to select performance measure
407
408     Returns
409     -----
410     C          -- float, optimal parameter value for an RBF-kernel
SVM
411     gamma     -- float, optimal parameter value for an RBF-kernel
SVM
412     """
413
414     print 'RBF SVM Hyperparameter Selection based on ' + str(metric)
+ ':'
415
416     ### ===== TODO : START ===== ###
417     # part 3b: create grid, then select optimal hyperparameters
using cross-validation
418
419     #rows are gamma and columns are C
420     performance_grid = np.zeros((5,4))
421     C_vals = [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
422     gamma_vals = [0.001, 0.01, 0.1, 1.0, 10]
423     for i in range(0,5):
424         for j in range(0,4):
425             performance_grid[i][j] =
cv_performance(SVC(kernel='rbf', gamma=gamma_vals[i], C=C_vals[j]),
426                 X, y, kf,
metric=metric)
427
428     indices = np.unravel_index(np.argmax(performance_grid),
performance_grid.shape)
429
430     return C_vals[indices[1]], gamma_vals[indices[0]],
performance_grid[indices]
431     ### ===== TODO : END ===== ###

```

```

432
433
434 def performance_CI(clf, X, y, metric="accuracy") :
435     """
436     Estimates the performance of the classifier using the 95% CI.
437
438     Parameters
439     -----
440         clf          -- classifier (instance of SVC or
DummyClassifier)
                        [already fit to data]
441         X            -- numpy array of shape (n,d), feature vectors
of test set
                        n = number of examples
                        d = number of features
442         y            -- numpy array of shape (n,), binary labels
{1,-1} of test set
443         metric       -- string, option used to select performance
measure
444
445     Returns
446     -----
447         score         -- float, classifier performance
448         lower         -- float, lower limit of confidence interval
449         upper         -- float, upper limit of confidence interval
450     """
451     n, d = X.shape
452     try :
453         y_pred = clf.decision_function(X)
454     except :
455         y_pred = clf.predict(X)
456     score = performance(y, y_pred, metric)
457
458     ### ===== TODO : START ===== ###
459     # part 4b: use bootstrapping to compute 95% confidence interval
460     # hint: use np.random.randint(...)
461     confidence_array = []
462     for t in range(0,1000):
463         bootstrapped_X = np.zeros((n,d))
464         bootstrapped_y = np.zeros(n)
465         bootstrapped_ypred = np.zeros(n)
466         avg = 0
467         for i in range(0,n):
468             index = np.random.randint(0, n)
469             bootstrapped_X[i,:]=X[index,:]
470             bootstrapped_y[i] = y[index]
471             bootstrapped_ypred[i] = y_pred[index]
472         confidence_array.append(performance(bootstrapped_y,
bootstrapped_ypred, metric))
473
474
475
476

```

```

477
478     confidence_array.sort()
479
480     return score, confidence_array[24], confidence_array[974]
481     ### ===== TODO : END ===== ###
482
483
484 #####
485 ##
486 # functions -- plotting
487 #####
488 ##
489
490 def lineplot(x, y, label):
491     """
492     Make a line plot.
493
494     Parameters
495     -----
496         x            -- list of doubles, x values
497         y            -- list of doubles, y values
498         label        -- string, label for legend
499     """
500     xx = range(len(x))
501     plt.plot(xx, y, linestyle='-', linewidth=2, label=label)
502     plt.xticks(xx, x)
503     plt.show()
504
505 def plot_results(metrics, classifiers, *args):
506     """
507     Make a results plot.
508
509     Parameters
510     -----
511         metrics      -- list of strings, metrics
512         classifiers   -- list of strings, classifiers
513         args         -- variable length argument
514                        results for baseline
515                        results for classifier 1
516                        results for classifier 2
517                        ...
518                        each results is a tuple (score, lower,
519 upper)
520     """
521     num_metrics = len(metrics)
522     num_classifiers = len(args) - 1
523

```

```

524     ind = np.arange(num_metrics) # the x locations for the groups
525     width = 0.7 / num_classifiers # the width of the bars
526
527     fig, ax = plt.subplots()
528
529     # loop through classifiers
530     rects_list = []
531     for i in xrange(num_classifiers):
532         results = args[i+1] # skip baseline
533         means = [it[0] for it in results]
534         errs = [(it[0] - it[1], it[2] - it[0]) for it in results]
535         rects = ax.bar(ind + i * width, means, width,
label=classifiers[i])
536         ax.errorbar(ind + i * width, means, yerr=np.array(errs).T,
fmt='none', ecolor='k')
537         rects_list.append(rects)
538
539     # baseline
540     results = args[0]
541     for i in xrange(num_metrics) :
542         mean = results[i][0]
543         err_low = results[i][1]
544         err_high = results[i][2]
545         xlim = (ind[i] - 0.8 * width, ind[i] + num_classifiers *
width - 0.2 * width)
546         plt.plot(xlim, [mean, mean], color='k', linestyle='-',
linewidth=2)
547         plt.plot(xlim, [err_low, err_low], color='k', linestyle='--
', linewidth=2)
548         plt.plot(xlim, [err_high, err_high], color='k', linestyle='--
-', linewidth=2)
549
550         ax.set_ylabel('Score')
551         ax.set_ylim(0, 1)
552         ax.set_xticks(ind + width / num_classifiers)
553         ax.set_xticklabels(metrics)
554         ax.legend()
555
556     def autolabel(rects):
557         """Attach a text label above each bar displaying its
height"""
558         for rect in rects:
559             height = rect.get_height()
560             ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                    '%.3f' % height, ha='center', va='bottom')
561
562         for rects in rects_list:
563             autolabel(rects)
564
565     plt.show()
566

```

```
567
568
569 #####
570 ##
571 # main
572 #####
573 ##
574
575 def main() :
576     # read the tweets and its labels
577     dictionary = extract_dictionary('../data/tweets.txt')
578     test_extract_dictionary(dictionary)
579     X = extract_feature_vectors('../data/tweets.txt', dictionary)
580     test_extract_feature_vectors(X)
581     y = read_vector_file('../data/labels.txt')
582
583     # shuffle data (since file has tweets ordered by movie)
584     X, y = shuffle(X, y, random_state=0)
585
586     # set random seed
587     np.random.seed(1234)
588
589     # split the data into training (training + cross-validation) and
590     # testing set
591     X_train, X_test = X[:560], X[560:]
592     y_train, y_test = y[:560], y[560:]
593
594     metric_list = ["accuracy", "f1_score", "auroc", "precision",
595                   "sensitivity", "specificity"]
596
597     ### ===== TODO : START ===== ###
598     test_performance()
599
600     # part 2b: create stratified folds (5-fold CV)
601     kf_strat = StratifiedKFold(n_splits=5)
602     cv_scores = cv_performance(SVC(), X_train, y_train, kf_strat)
603     print "scores for CV: " + str(cv_scores)
604
605     # part 2c: finding the optimal C
606     best_cs = []
607     for metric in metric_list:
608         best_cs.append(select_param_linear(X, y, kf_strat,
609                                           metric=metric, plot=False))
610     print best_cs
611
612     # part 2d: for each metric, select optimal hyperparameter for
613     # linear-kernel SVM using CV
614     # plot the metrics
615     plot_metric_2d(X, y, kf_strat)
616
617     # part 3c: for each metric, select o
```

```
611     # optimal hyperparameter for RBF-SVM using CV
612     C, gamma, score = select_param_rbf(X, y, kf_strat)
613     print "optimal C for accuracy= %f, optimal gamma for accuracy
%f, score was %f" % (C, gamma, score)
614
615
616
617     metrics = ["accuracy", "f1_score", "auroc", "precision",
"sensitivity", "specificity"]
618     for metric in metrics:
619         #C, gamma, score = select_param_rbf(X, y, kf_strat,
metric=metric)
620         score = cv_performance(SVC(kernel='rbf', gamma=0.01,
C=10.0),
X, y, kf_strat,
metric=metric)
621
622         print "optimal C for %s= %f, optimal gamma %f, score was %f"
% (metric, 10.0, 0.01, score)
623
624
625
626
627
628     # part 4a: train linear- and RBF-kernel SVMs with selected
hyperparameters
629     linear_svm = SVC(kernel='linear', C=1)
630     rbf_svm = SVC(kernel='rbf', gamma=0.01, C=10.0)
631     dummy_classifier = DummyClassifier(strategy="most_frequent")
632     linear_svm.fit(X_train,y_train)
633     rbf_svm.fit(X_train,y_train)
634     dummy_classifier.fit(X_train, y_train)
635
636     # part 4c: use bootstrapping to report performance on test data
637     # use plot_results(...) to make plot
638     linear_svm_performance = []
639     print "LINEAR SVM"
640     for metric in metrics:
641         result_metric = performance_CI(linear_svm, X_test, y_test,
metric= metric)
642         linear_svm_performance.append(result_metric)
643
644         print "for %s: score = %f, low end: %f, high end = %f" %
(metric, result_metric[0],
result_metric[1], result_metric[2])
645
646
647     rbf_svm_performance = []
648     print "RBF SVM"
649     for metric in metrics:
650         result_metric = performance_CI(rbf_svm, X_test, y_test,
```

```
metric= metric)
651     rbf_svm_performance.append(result_metric)
652     print "for %s: score = %f, low end: %f, high end = %f" %
(metric, result_metric[0],
653     result_metric[1], result_metric[2])
654
655     # use a baseline performance classifier for comparison
656     dummy_classifier_performance = []
657     print "Baseline classifier - majority classifier with Dummy
Classifier"
658     for metric in metrics:
659         result_metric = performance_CI(dummy_classifier, X_test,
y_test, metric= metric)
660         dummy_classifier_performance.append(result_metric)
661         print "for %s: score = %f, low end: %f, high end = %f" %
(metric, result_metric[0],
662         result_metric[1], result_metric[2])
663
664
665
666     # create the bar plot
667     classifiers = ["Linear SVM", "RBF SVM"]
668     plot_results(metrics, classifiers, dummy_classifier_performance,
linear_svm_performance, rbf_svm_performance)
669
670
671     # part 5: identify important features
672     full_linear_svm = SVC(kernel='linear', C=1)
673     full_linear_svm.fit(X,y)
674
675     print "positive linear coefs: "
676
677     coefs_index = full_linear_svm.coef_[0].argsort()[-10:][::-1]
678     for coef in coefs_index:
679         print full_linear_svm.coef_[0][coef]
680         for word, index in dictionary.iteritems():
681             if index == coef:
682                 print word
683
684
685
686     print "negative linear coefs: "
687     coefs_index = full_linear_svm.coef_[0].argsort()[ :10] [::-1]
688     for coef in coefs_index:
689         print full_linear_svm.coef_[0][coef]
690         for word, index in dictionary.iteritems():
691             if index == coef:
692                 print word
```



```
693
694     ### ===== TODO : END ===== ###
695
696     ### ===== TODO : START ===== ###
697     # Twitter contest
698     # uncomment out the following, and be sure to change the
filename
699     """
700     X_held = extract_feature_vectors('../data/held_out_tweets.txt',
dictionary)
701     # your code here
702     # y_pred = best_clf.decision_function(X_held)
703     write_label_answer(y_pred, '../data/yjw_twitter.txt')
704     """
705     ### ===== TODO : END ===== ###
706
707
708 if __name__ == "__main__" :
709     main()
```