MA598 HW 2

Guangtong Shen (0028213116)

Fall 2016

Task

use Sklearn.svm.svc to study handwritten digits from the processed US Postal Service Zip Code data set. Download the data (the format of each row is: digit intensity symmetry) for training and testing:

http://www.amlbook.com/data/zip/features.train

http://www.amlbook.com/data/zip/features.test

We will train a one-versus-one (one digit is class +1 and another digit is class -1) classifier for the digits '1' (+1) and '5' (-1).

Definition

- Ein returns the in training sample error of the current svm model. It is the fraction of in training sample points which got misclassified.
- Eout returns the testing sample error of the current svm model. It is the fraction of testing sample points which got misclassified.
- Ecv returns the leave one out cross validation in training sample error of the current svm model.
- accuracy over the testing set = 1 - Eout

Import data

Read train and test dataset, only keep the rows of digit 1 (labeled 1) and digit 5 (labeled -1).

```
# import data

import random
import numpy as np
```

```
def parseDataWithShuffle(filename):
    x = []
    y = []
    xy = []
    with open(filename) as file:
        for eachline in file:
            line = eachline.strip().split('  ')
            a,b,c = (int(float(line[0])), float(line[1]), float(line[2]))
            if(a == 1 or a == 5):
                xy.append([b, c, 1 if a == 1 else -1])

    random.shuffle(xy)
    x = [[row[0], row[1]] for row in xy]
    y = [row[2] for row in xy]
    count1 = len([yi for yi in y if yi == 1])
    print "1 appears " + str(count1) + " times in " + filename;
    count5 = len([yi for yi in y if yi == 0])
    print "5 appears " + str(count5) + " times in " + filename;
    return  np.array(x), np.array(y)

x_train, y_train = parseDataWithShuffle("features.train")
x_test, y_test = parseDataWithShuffle("features.test")
```

```
1 appears 1005 times in features.train
5 appears 0 times in features.train
1 appears 264 times in features.test
5 appears 0 times in features.test
```

Question A

- Consider the linear kernel K(x_n, x_m) = x_n^T x_m. Train and test using all of the points, writing the output to an output file hw2.txt.

```
from sklearn.svm import SVC
import matplotlib.pyplot as plt

svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(x_train, y_train)
y_predict = svm.predict(x_test)

with open('hw2.txt', 'w') as file:
    file.write("Test set labels \t Predicted labels\n")
    for i,j in zip(y_test, y_predict):
        file.write(str(i) + '\t' + str(j) + '\n')

def countError(y_real,y_pred):
    if len(y_real) != len(y_pred):
        return 0
    errors = [1 for i,j in zip(y_real, y_pred) if i != j]
    return float(len(errors))/len(y_real)
```

```
Ein  = countError(y_train, svm.predict(x_train))
Eout = countError(y_test, svm.predict(x_test))

print "Accuracy over testing set: " + str(1-Eout)
print "number of support vectors for each class: " + str(svm.n_support_)
```

```
Accuracy over testing set: 0.978773584906
number of support vectors for each class: [14 14]
```

- In addition to using all of the training examples, try subsets of the training data and print out accuracy over the testing set (1 - Eout (over all test examples), and the number of support vectors. Try with the first {50, 100, 200, 800} points with the linear kernel. The output of these experiments should be written in Markdown cells.

```
svm.fit(x_train[0:50], y_train[0:50])
Eout = countError(y_test, svm.predict(x_test))
print "Training set size: 50"
print "Accuracy over testing set: " + str(1-Eout)
print "number of support vectors for each class: " + str(svm.n_support_)

svm.fit(x_train[0:100], y_train[0:100])
Eout = countError(y_test, svm.predict(x_test))
print "Training set size: 100"
print "Accuracy over testing set: " + str(1-Eout)
print "number of support vectors for each class: " + str(svm.n_support_)

svm.fit(x_train[0:200], y_train[0:200])
Eout = countError(y_test, svm.predict(x_test))
print "Training set size: 200"
print "Accuracy over testing set: " + str(1-Eout)
print "number of support vectors for each class: " + str(svm.n_support_)

svm.fit(x_train[0:800], y_train[0:800])
Eout = countError(y_test, svm.predict(x_test))
print "Training set size: 800"
print "Accuracy over testing set: " + str(1-Eout)
print "number of support vectors for each class: " + str(svm.n_support_)
```

```
Training set size: 50
Accuracy over testing set: 0.97641509434
number of support vectors for each class: [3 3]
Training set size: 100
Accuracy over testing set: 0.974056603774
number of support vectors for each class: [3 3]
Training set size: 200
Accuracy over testing set: 0.978773584906
number of support vectors for each class: [4 4]
Training set size: 800
```

```
Accuracy over testing set: 0.978773584906
number of support vectors for each class: [7 7]
```

Question B

- Consider the polynomial kernel K(x_n, x_m) = (1 + x_n^T x_m)^Q, where Q is the degree of the polynomial.
- Comparing Q = 2 with Q = 5, which of the following statements is correct?
  i. When C = 0.0001, Ein is higher at Q = 5.
  ii. When C = 0.001, the number of support vectors is lower at Q = 5.
  iii. When C = 0.01, Ein is higher at Q = 5.
  iv. When C = 1, Eout is lower at Q = 5.
  v. None of the above

```
template = 'degree = {0} \t Ein = {1:10.9f} \t Eout = {2:10.9f} \t NumSV = {3}';

for c_candidate in [0.0001, 0.001, 0.01, 1.0]:
    print '\nC = ' + str(c_candidate);
    for d_candidate in [2,5]:
        svm = SVC(kernel='poly', C=c_candidate, degree = d_candidate, random_state=0)
        svm.fit(x_train, y_train)
        Ein = countError(y_train, svm.predict(x_train));
        Eout = countError(y_test, svm.predict(x_test));
        NumSv = svm.n_support_[0]
        print template.format(d_candidate, Ein, Eout, NumSv)
```

```
C = 0.0001
degree = 2    Ein = 0.022421525    Eout = 0.030660377      NumSV = 255
degree = 5    Ein = 0.006406150    Eout = 0.018867925      NumSV = 21

C = 0.001
degree = 2    Ein = 0.007046765    Eout = 0.018867925      NumSV = 76
degree = 5    Ein = 0.005124920    Eout = 0.016509434      NumSV = 14

C = 0.01
degree = 2    Ein = 0.004484305    Eout = 0.018867925      NumSV = 27
degree = 5    Ein = 0.004484305    Eout = 0.016509434      NumSV = 13

C = 1.0
degree = 2    Ein = 0.004484305    Eout = 0.018867925      NumSV = 13
degree = 5    Ein = 0.004484305    Eout = 0.016509434      NumSV = 13
```

From the result above, these statements are right:

```
2. When C = 0.001, the number of support vectors is lower at Q = 5.
4. When C = 1, Eout is lower at Q = 5.
```

Question C

- Consider the 1 versus 5 classifier with Q = 2 and C ∈ {0.001, 0.01, 0.1, 1}.
- Which of the following statements is correct? Going up or down means strictly so.

  [a] The number of support vectors goes down when C goes up.
  [b] The number of support vectors goes up when C goes up.
  [c] Eout goes down when C goes up.
  [d] Maximum C achieves the lowest Ein.
  [e] None of the above

```
for c_candidate in [0.001, 0.01, 0.1, 1.0]:
    print '\nC = ' + str(c_candidate);
    svm = SVC(kernel='poly', C=c_candidate, degree = 2, random_state=0)
    svm.fit(x_train, y_train)
    Ein = countError(y_train, svm.predict(x_train));
    Eout = countError(y_test, svm.predict(x_test));
    NumSv = svm.n_support_[0]

    print template.format(2, Ein, Eout, NumSv)
```

```
C = 0.001
degree = 2   Ein = 0.007046765   Eout = 0.018867925      NumSV = 76

C = 0.01
degree = 2   Ein = 0.004484305   Eout = 0.018867925      NumSV = 27

C = 0.1
degree = 2   Ein = 0.004484305   Eout = 0.018867925      NumSV = 14

C = 1.0
degree = 2   Ein = 0.004484305   Eout = 0.018867925      NumSV = 13
```

From the above result, these statements are right:
[a] The number of support vectors goes down when C goes up.

Cross Validation

In the next two problems, we will experiment with 10-fold cross validation for the polynomial kernel. Because Ecv is a random variable that depends on the random partition of the data, we will try100 runs with different partitions and base our answer on how many runs lead to a particular choice.

Question D

- Consider the 1 versus 5 classifier with Q = 2. We use Ecv to select C ∈ {0.0001, 0.001, 0.01, 0.1, 1}. If there is a tie in Ecv, select the smaller C.
- Within the 100 random runs, which of the following statements is correct?

  [a] C = 0.0001 is selected most often.
  [b] C = 0.001 is selected most often.
  [c] C = 0.01 is selected most often.
  [d] C = 0.1 is selected most often.
  [e] C = 1 is selected most often.

```
from sklearn.cross_validation import StratifiedKFold

def calcEcv(X,Y,c_candidate, random): # small C to tolerate error thus to avoid overfitting
    EcvList = []
    svm = SVC(kernel='poly', C=c_candidate, degree = 2, random_state=random)
    kfold = StratifiedKFold(y=Y, n_folds=10, shuffle=True, random_state=random)
    for k, (train, test) in enumerate(kfold):
        svm.fit(X[train], Y[train])
        Ecv = countError(Y[test], svm.predict(X[test]))
        EcvList.append(Ecv)
    return sum(EcvList)/len(EcvList)

C_appear_times = {}
C_candidates = [0.0001, 0.001, 0.01, 0.1, 1]
for C in C_candidates:
    C_appear_times[C] = 0

for i in xrange(100):
    minEcv = 1.0
    C_selection = 0.0001
    EcvOfCandidates = [calcEcv(x_train, y_train, c, i) for c in C_candidates]
    C_selection = C_candidates[np.argmin(EcvOfCandidates)]
    C_appear_times[C_selection] = C_appear_times[C_selection] + 1

print C_appear_times
```

```
{1: 13, 0.001: 0, 0.0001: 0, 0.1: 4, 0.01: 83}
```

The minimun Ecv is obtained when C=0.01 for 83 times
So the answer is

```
[c] C = 0.01 is selected most often.
```

Question E

- Again, consider the 1 versus 5 classifier with Q = 2.
- For the winning selection in the previous problem, the average value of Ecv over the 100 runs is closest to
  [a] 0.001
  [b] 0.003
  [c] 0.005
  [d] 0.007
  [e] 0.009

```
EcvList = [calcEcv(x_train, y_train, 0.01, i) for i in range(100)]
avgEcv = sum(EcvList)/len(EcvList)
print avgEcv
```

```
0.00449075827262
```

So the average Ecv is closet to

```
[c] 0.005
```

Questiion F

- Consider the radial basis function (RBF) kernel K(x_n, x_m) = e^(- ||xn - xm||^2) in the SVC approach.
- Which value of C ∈ {0.01, 1, 100, 10^4, 10^6} results in the lowest Ein? The lowest Eout?

```
template = 'Ein = {0:10.9f} \t Eout = {1:10.9f} \t NumSV = {2}';

for c_candidate in [0.01, 1, 100, 1.0e4, 1.0e6]:
    print '\nC = ' + str(c_candidate);
    svm = SVC(kernel='rbf', C=c_candidate, random_state=0)
    svm.fit(x_train, y_train)
    Ein = countError(y_train, svm.predict(x_train));
    Eout = countError(y_test, svm.predict(x_test));
    NumSv = svm.n_support_[0]
    print template.format(Ein, Eout, NumSv)
```

```
C = 0.01
Ein = 0.003843690     Eout = 0.021226415      NumSV = 174

C = 1
Ein = 0.004484305     Eout = 0.021226415      NumSV = 18
```

```
C = 100
Ein = 0.003203075    Eout = 0.018867925      NumSV = 11

C = 10000.0
Ein = 0.002562460    Eout = 0.018867925      NumSV = 10

C = 1000000.0
Ein = 0.001281230    Eout = 0.021226415      NumSV = 11
```

Lowest Ein comes from C = 10^6

Lowest Eout comes from C = 100

The larger C is, the more accurate and more likely to overfit is the model.