```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.utils import shuffle
from collections import Counter
import matplotlib.pyplot as plt

indata = np.loadtxt(r"C:\Users\Zhenc_000\Documents\Caltech\CS156\Set8\features.train.txt")
outdata = np.loadtxt(r"C:\Users\Zhenc_000\Documents\Caltech\CS156\Set8\features.test.txt")

def length(number, source): #function to calculate number of data points for each digit
    return sum(1 for i in range (len(source)) if source[i,0] == number)

def populate(number, source): #create x-vector for a single digit
    array = np.zeros([length(number,source),2])
    array[:,0] = [source[i,1] for i in range(len(source)) if source[i,0] == number]
    array[:,1] = [source[i,2] for i in range(len(source)) if source[i,0] == number]
    return array

train_set = [populate(x, indata) for x in range(10)]
test_set = [populate(x, outdata) for x in range(10)]

def populate_all(number, source): #set classification of all other digits to -1 and target
digit to 1
    array = np.array([1 if source[i,0] == number else -1 for i in range(len(source))])
    return array

train_all = [populate_all(x, indata) for x in range(10)]
test_all = [populate_all(x, outdata) for x in range(10)]

errors = []

for i in range(10): #find E_in for each digit
    clf = SVC(0.01, 'poly', 2, 1, 1) #find the SVC
    clf.fit(indata[:,1:3],train_all[i])

    errors.append((len(train_all[i])-np.dot(np.sign(clf.predict(indata[:,1:3])),train_all[i]))/(2
    *len(train_all[i])))

plt.plot(errors) #plot E_in
plt.xlabel('Digit')
plt.ylabel('In-sample error')
plt.show()

clf = SVC(0.01, 'poly', 2, 1, 1) #find number of support vectors for digits 0 and 1
clf.fit(indata[:,1:3],train_all[0])
print(sum(clf.n_support_))

train15 = np.append(train_set[1],train_set[5],axis=0) #build set for digits 1 and 5 (ovo)
y15in = np.append(np.ones([len(train_set[1])]),-1*np.ones([len(train_set[5])]),axis=0)
test15 = np.append(test_set[1],test_set[5],axis=0)
y15out = np.append(np.ones([len(test_set[1])]),-1*np.ones([len(test_set[5])]),axis=0)
```

```
support = []
e_in = []
e_out = []

for i in range(4): #test different results for Q = 2
    clf = SVC(10**(-i), 'poly', 2, 1, 1)
    clf.fit(train15,y15in)
    support.insert(0,sum(clf.n_support_))
    e_in.insert(0,(len(train15)-np.dot(np.sign(clf.predict(train15)),y15in))/(2*len(train15)))
    e_out.insert(0,(len(test15)-np.dot(np.sign(clf.predict(test15)),y15out))/(2*len(test15)))

for i in range(4): #test different results for Q = 5
    clf = SVC(10**(-i), 'poly', 5, 1, 1)
    clf.fit(train15,y15in)
    support.insert(0,sum(clf.n_support_))
    e_in.insert(0,(len(train15)-np.dot(np.sign(clf.predict(train15)),y15in))/(2*len(train15)))
    e_out.insert(0,(len(test15)-np.dot(np.sign(clf.predict(test15)),y15out))/(2*len(test15)))

print(support)
print(e_in)
print(e_out)

best = np.ones(1000)
scores = []

for i in range(1000):
    score = 0
    x_train, y_train = shuffle(train15, y15in) #rearrange the training set for each iteration
    clf = SVC(10**(-3), 'poly', 2, 1, 1).fit(train15, y15in) #find the E_cv of winning C value
    scores.append(1-sum(cross_val_score(clf, x_train, y_train, cv=10))/10)
    for j in range(5): #compute E_cv for each C value
        clf = SVC(10**(-j), 'poly', 2, 1, 1).fit(train15, y15in)
        if sum(cross_val_score(clf, x_train, y_train, cv=10)) >= score:
            best[i] = j
            score = sum(cross_val_score(clf, x_train, y_train, cv=10))

print(Counter(best))
print(sum(scores)/100)

for i in range(-2,7): #determine the value of C that minimizes E_in and E_out
    clf = SVC(10**(i), 'rbf', gamma=1).fit(train15, y15in)
    e_in.append(1-clf.score(train15,y15in))
    e_out.append(1-clf.score(test15,y15out))

print(e_in)
print(e_out)
```