CS/CNS/EE 156a

Homework 8

Sung Hoon Choi

(Last name: Choi)

1. Answer: [d]

Constrained Optimization: Minimize $\frac{1}{2}w^Tw$ subject to $y_n(w^Tx_n + b) \geq 1$ for n=1,2,…,N. Since the data sets are given, the variables are just w and b. Since d is the dimensionality of the input space, *w* gives d variables. Besides, we have *b* as another variable. Therefore, it is a quadratic programming problem with d+1 variables in total.

2. Answer: [a]

0 versus all: 0.105884 **(the highest)**
2 versus all: 0.100261
4 versus all: 0.089425
6 versus all: 0.091071
8 versus all: 0.074338
Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 2

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_all_label(data_array, digit):
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit:
            labelled_data_array.append(1)
        else:
            labelled_data_array.append(-1)
    return labelled_data_array

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)

label_0_vs_all = one_vs_all_label(train_data_array_np,0) #label +1 or -1
label_2_vs_all = one_vs_all_label(train_data_array_np,2)
label_4_vs_all = one_vs_all_label(train_data_array_np,4)
label_6_vs_all = one_vs_all_label(train_data_array_np,6)
label_8_vs_all = one_vs_all_label(train_data_array_np,8)
```

```
clf_digit_0 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_2 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_4 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_6 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_8 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)

clf_digit_0.fit(train_data_array_np[:,1:],label_0_vs_all) #fit the SVM
clf_digit_2.fit(train_data_array_np[:,1:],label_2_vs_all)
clf_digit_4.fit(train_data_array_np[:,1:],label_4_vs_all)
clf_digit_6.fit(train_data_array_np[:,1:],label_6_vs_all)
clf_digit_8.fit(train_data_array_np[:,1:],label_8_vs_all)

label_0_predict = clf_digit_0.predict(train_data_array_np[:,1:]) #predict the output using SVM
label_2_predict = clf_digit_2.predict(train_data_array_np[:,1:])
label_4_predict = clf_digit_4.predict(train_data_array_np[:,1:])
label_6_predict = clf_digit_6.predict(train_data_array_np[:,1:])
label_8_predict = clf_digit_8.predict(train_data_array_np[:,1:])

print("0 versus all: %2f\n" %calculate_binary_error(label_0_predict, label_0_vs_all)) #calculate Ein
print("2 versus all: %2f\n" %calculate_binary_error(label_2_predict, label_2_vs_all))
print("4 versus all: %2f\n" %calculate_binary_error(label_4_predict, label_4_vs_all))
print("6 versus all: %2f\n" %calculate_binary_error(label_6_predict, label_6_vs_all))
print("8 versus all: %2f\n" %calculate_binary_error(label_8_predict, label_8_vs_all))
```

3. Answer: [a]

1 versus all: 0.014401 **(the lowest)**

3 versus all: 0.090248

5 versus all: 0.076258

7 versus all: 0.088465

9 versus all: 0.088328

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 3

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_all_label(data_array, digit):
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit:
            labelled_data_array.append(1)
        else:
            labelled_data_array.append(-1)
    return labelled_data_array

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)

label_1_vs_all = one_vs_all_label(train_data_array_np,1) #label +1 or -1
label_3_vs_all = one_vs_all_label(train_data_array_np,3)
label_5_vs_all = one_vs_all_label(train_data_array_np,5)
label_7_vs_all = one_vs_all_label(train_data_array_np,7)
```

```
label_9_vs_all = one_vs_all_label(train_data_array_np,9)

clf_digit_1 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_3 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_5 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_7 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
clf_digit_9 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)

clf_digit_1.fit(train_data_array_np[:,1:],label_1_vs_all) #fit the SVM
clf_digit_3.fit(train_data_array_np[:,1:],label_3_vs_all)
clf_digit_5.fit(train_data_array_np[:,1:],label_5_vs_all)
clf_digit_7.fit(train_data_array_np[:,1:],label_7_vs_all)
clf_digit_9.fit(train_data_array_np[:,1:],label_9_vs_all)

label_1_predict = clf_digit_1.predict(train_data_array_np[:,1:]) #predict the output using SVM
label_3_predict = clf_digit_3.predict(train_data_array_np[:,1:])
label_5_predict = clf_digit_5.predict(train_data_array_np[:,1:])
label_7_predict = clf_digit_7.predict(train_data_array_np[:,1:])
label_9_predict = clf_digit_9.predict(train_data_array_np[:,1:])

print("1 versus all: %2f\n" %calculate_binary_error(label_1_predict, label_1_vs_all)) #calculate Ein
print("3 versus all: %2f\n" %calculate_binary_error(label_3_predict, label_3_vs_all))
print("5 versus all: %2f\n" %calculate_binary_error(label_5_predict, label_5_vs_all))
print("7 versus all: %2f\n" %calculate_binary_error(label_7_predict, label_7_vs_all))
print("9 versus all: %2f\n" %calculate_binary_error(label_9_predict, label_9_vs_all))
```

4. Answer: [c]

The number of support vectors for 0 versus all: 2179

The number of support vectors for 1 versus all: 386

Difference: 1793

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 4

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_all_label(data_array, digit):
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit:
            labelled_data_array.append(1)
        else:
            labelled_data_array.append(-1)
    return labelled_data_array

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)

label_0_vs_all = one_vs_all_label(train_data_array_np,0) #label +1 or -1
label_1_vs_all = one_vs_all_label(train_data_array_np,1)

clf_digit_0 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0)
```

```
clf_digit_0.fit(train_data_array_np[:,1:],label_0_vs_all) #fit the SVM
clf_digit_1.fit(train_data_array_np[:,1:],label_1_vs_all)

print("Number of support vectors: label 0: ", len(clf_digit_0.support_)) #Number of support vectors
print("Number of support vectors: label 1: ", len(clf_digit_1.support_))
print("Difference: ", len(clf_digit_0.support_)-len(clf_digit_1.support_))
```

5. Answer: [d]

The simulation output is as follows.

```
C=0.001 ----------------------------------------
Number of support vectors: 76
Ein: 0.004484
Eout: 0.016509
C=0.01 -----------------------------------------
Number of support vectors: 34
Ein: 0.004484
Eout: 0.018868
C=0.1 ------------------------------------------
Number of support vectors: 24
Ein: 0.004484
Eout: 0.018868
C=1 --------------------------------------------
Number of support vectors: 24
Ein: 0.003203
Eout: 0.018868
------------------------------------------------
```

As we can see from the simulation result, when C goes up, although the number of support vectors decreases in the interval of C=0.001 to C=0.1, the number of support vectors stays same in the interval of C=0.1 to C=1. Also, when C goes up, although E_{out} increases in the interval of C=0.001 to C=0.01, it stays constant in the interval of C=0.01 to C=1. Finally, we can see that E_{in} is lowest when C=1. Therefore, the only correct answer is [d].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 5

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5 and 6.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5 and 6. (one versus one)
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)

print("C=0.001 ------------------------------------")
clf_digit_1_and_5 = SVC(C=0.001, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=0.01 -------------------------------------")
clf_digit_1_and_5 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=0.1 --------------------------------------")
clf_digit_1_and_5 = SVC(C=0.1, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=1 ----------------------------------------")
clf_digit_1_and_5 = SVC(C=1, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("--------------------------------------------")
```

6. Answer: [b]

The simulation output (result) is as follows:

```
------------------------C=0.0001 ------------------------
-----Q=2-----
Number of support vectors: 236
Ein: 0.008969
Eout: 0.016509
-----Q=5-----
Number of support vectors: 26
```

```
Ein: 0.004484
Eout: 0.018868
-----------------------C=0.001 ------------------------
-----Q=2-----
Number of support vectors: 76
Ein: 0.004484
Eout: 0.016509
-----Q=5-----
Number of support vectors: 25
Ein: 0.004484
Eout: 0.021226
------------------------C=0.01 -------------------------
-----Q=2-----
Number of support vectors: 34
Ein: 0.004484
Eout: 0.018868
-----Q=5-----
Number of support vectors: 23
Ein: 0.003844
Eout: 0.021226
------------------------C=1 ---------------------------
-----Q=2-----
Number of support vectors: 24
Ein: 0.003203
Eout: 0.018868
-----Q=5-----
Number of support vectors: 21
Ein: 0.003203
Eout: 0.021226
-------------------------------------------------------
```

As the simulation output describes, when C=0.0001, E_{in} is higher at Q=2. When C=0.001, the number of support vectors is lower at Q=5. When C=0.01, E_{in} is higher at Q=2. When C=1, E_{out} is lower at Q=2. Therefore, the only correct choice is [b].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 6

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5 and 6.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5 and 6. (one versus one)
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)

print("------------------------C=0.0001 ------------------------")
print("-----Q=2-----")
clf_digit_1_and_5 = SVC(C=0.0001, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("-----Q=5-----")
clf_digit_1_and_5 = SVC(C=0.0001, kernel='poly', degree=5, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("------------------------C=0.001 -------------------------")
print("-----Q=2-----")
clf_digit_1_and_5 = SVC(C=0.001, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("-----Q=5-----")
clf_digit_1_and_5 = SVC(C=0.001, kernel='poly', degree=5, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("------------------------C=0.01 --------------------------")
print("-----Q=2-----")
clf_digit_1_and_5 = SVC(C=0.01, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout
print("-----Q=5-----")
clf_digit_1_and_5 = SVC(C=0.01, kernel='poly', degree=5, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("------------------------C=1 ----------------------------")
print("-----Q=2-----")
clf_digit_1_and_5 = SVC(C=1, kernel='poly', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("-----Q=5-----")
clf_digit_1_and_5 = SVC(C=1, kernel='poly', degree=5, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout
print("----------------------------------------------------------")
```

7. Answer: [b]

The simulation output is as follows:

C=0.0001 chosen: 0

C=0.001 chosen: 58 **(selected most often)**

C=0.01 chosen: 20

C=0.1 chosen: 10

C=1 chosen: 12

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 7

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5,6,7,8.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5,6,7,8.
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

def divide_into_partitions(data, folds):
    partition_size = len(data)//folds
    partitions = []
    for i in range(0, folds):
        partitions.append(data[i*partition_size:(i+1)*partition_size])
    return partitions                  #partition[0]: first partition
                                     #partition[1]: second partition
                                     # . . .

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)

Choice_a = 0  #Number of runs for the case when choice [a] is chosen.
Choice_b = 0  #Number of runs for the case when choice [b] is chosen.
Choice_c = 0  #Number of runs for the case when choice [c] is chosen.
Choice_d = 0  #Number of runs for the case when choice [d] is chosen.
Choice_e = 0  #Number of runs for the case when choice [e] is chosen.
for run in range (0,100):
    Total_Run_Error = 0
    np.random.shuffle((filtered_data_1_vs_5_train))
    partitions = np.array(divide_into_partitions(filtered_data_1_vs_5_train,10))
    partitions_labels = np.array(divide_into_partitions(one_vs_one_label(filtered_data_1_vs_5_train,1,5),10))
    Error = [0, 0, 0, 0, 0]
    for C_value in (0.0001, 0.001, 0.01, 0.1, 1):
        clf_digit_1_and_5 = SVC(C=C_value, kernel='poly', degree=2, coef0=1.0, gamma=1.0)  # kernel definition
        Each_CV_Error = 0
        for i in range (0,len(partitions)):
            cv_training_partitions = np.delete(partitions,i,0) #Leave on partition for validation.
            cv_training_labels = np.delete(partitions_labels,i,0)
            concat_cv_training_partitions = cv_training_partitions[0]
            concat_cv_training_labels = cv_training_labels[0]
            for j in range (1, len(cv_training_partitions)): #Need to reshape the partitions for processing.
                concat_cv_training_partitions = np.concatenate((concat_cv_training_partitions,cv_training_partitions[j]))
                concat_cv_training_labels = np.concatenate((concat_cv_training_labels,cv_training_labels[j]))
            clf_digit_1_and_5.fit(concat_cv_training_partitions[:,1:],concat_cv_training_labels) #Train
            predict = clf_digit_1_and_5.predict(partitions[i][:,1:]) #Validate
            Each_CV_Error = Each_CV_Error + calculate_binary_error(predict, partitions_labels[i]) #Get the error
        if C_value == 0.0001:
            Error[0] = Each_CV_Error/len(partitions)
            #print("a:", Error[0])
        elif C_value == 0.001:
            Error[1] = Each_CV_Error/len(partitions)
            #print("b:",Error[1])
        elif C_value == 0.01:
            Error[2] = Each_CV_Error/len(partitions)
            #print("c:",Error[2])
        elif C_value == 0.1:
            Error[3] = Each_CV_Error/len(partitions)
            #print("d:",Error[3])
        elif C_value == 1:
            Error[4] = Each_CV_Error/len(partitions)
            #print("e:",Error[4])

    if np.argmin(Error) == 0:       #Select the C depending on the error.
        Choice_a = Choice_a + 1
    elif np.argmin(Error) == 1:
        Choice_b = Choice_b + 1
    elif np.argmin(Error) == 2:
        Choice_c = Choice_c + 1
    elif np.argmin(Error) == 3:
        Choice_d = Choice_d + 1
    elif np.argmin(Error) == 4:
        Choice_e = Choice_e + 1
    Total_Run_Error = Total_Run_Error + Each_CV_Error

#Problem 7
print("C=0.0001 chosen:", Choice_a)
print("C=0.001 chosen:", Choice_b)
print("C=0.01 chosen:", Choice_c)
print("C=0.1 chosen:", Choice_d)
print("C=1 chosen:", Choice_e)
```

8. Answer: [c]

From Problem 7, we found that the winning selection is C=0.001.

By running the experiment 100 times with C=0.001, I found the average value of E_{cv} to be 0.00476.

Therefore, its closest value is 0.005.

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 8

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5,6,7,8.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5,6,7,8.
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

def divide_into_partitions(data, folds):
    partition_size = len(data)//folds
    partitions = []
    for i in range(0, folds):
        partitions.append(data[i*partition_size:(i+1)*partition_size])
    return partitions                 #partition[0]: first partition
                                    #partition[1]: second partition
                                    # . . .

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)
```

```
Total_Run_Error = 0
for run in range (0,100):
    np.random.shuffle((filtered_data_1_vs_5_train))
    partitions = np.array(divide_into_partitions(filtered_data_1_vs_5_train,10))
    partitions_labels = np.array(divide_into_partitions(one_vs_one_label(filtered_data_1_vs_5_train,1,5),10))

    clf_digit_1_and_5 = SVC(C=0.001, kernel='poly', degree=2, coef0=1.0, gamma=1.0)  # kernel definition
    Each_CV_Error = 0
    for i in range (0,len(partitions)):
        cv_training_partitions = np.delete(partitions,i,0) #Leave on partition for validation.
        cv_training_labels = np.delete(partitions_labels,i,0)
        concat_cv_training_partitions = cv_training_partitions[0]
        concat_cv_training_labels = cv_training_labels[0]
        for j in range (1, len(cv_training_partitions)): #Need to reshape the partitions for processing.
            concat_cv_training_partitions = np.concatenate((concat_cv_training_partitions,cv_training_partitions[j]))
            concat_cv_training_labels = np.concatenate((concat_cv_training_labels,cv_training_labels[j]))
        clf_digit_1_and_5.fit(concat_cv_training_partitions[:,1:],concat_cv_training_labels) #Train
        predict = clf_digit_1_and_5.predict(partitions[i][:,1:]) #Validate
        Each_CV_Error = Each_CV_Error + calculate_binary_error(predict, partitions_labels[i]) #Get the error
    Each_Run_Error = Each_CV_Error/len(partitions)
    Total_Run_Error = Total_Run_Error + Each_Run_Error

print("Average Ecv for C=0.001:", Total_Run_Error/100) #Problem 8
```

9. Answer: [e]

The simulation result is as follows:

C=0.01 -----------------------------------

Number of support vectors: 406

Ein: 0.003844

Eout: 0.023585

C=1 ------------------------------------

Number of support vectors: 31

Ein: 0.004484

Eout: 0.021226

C=100 ------------------------------------

Number of support vectors: 22

Ein: 0.003203

Eout: 0.018868

C=10^4 ------------------------------------

Number of support vectors: 19

Ein: 0.002562

Eout: 0.023585

C=10^6 ------------------------------------

Number of support vectors: 17

Ein: 0.000641 **(The lowest E_{in})**

Eout: 0.023585

Therefore, C=10^6 results in the lowest E_{in}.

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 9 & Problem 10

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5 and 6.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5 and 6. (one versus one)
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)

print("C=0.01 -------------------------------------")
clf_digit_1_and_5 = SVC(C=0.01, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=1 --------------------------------------")
clf_digit_1_and_5 = SVC(C=1, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=100 ---------------------------------------")
clf_digit_1_and_5 = SVC(C=100, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=10^4 -----------------------------------------")
clf_digit_1_and_5 = SVC(C=10000, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=10^6 -----------------------------------------")
clf_digit_1_and_5 = SVC(C=1000000, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
```

```
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("----------------------------------------------")
```

10. Answer: [c]

Again, the simulation result is as follows:

C=0.01 -----------------------------------

Number of support vectors: 406

Ein: 0.003844

Eout: 0.023585

C=1 -------------------------------------

Number of support vectors: 31

Ein: 0.004484

Eout: 0.021226

C=100 -------------------------------------

Number of support vectors: 22

Ein: 0.003203

Eout: 0.018868 **(The lowest E_{out})**

C=10^4 ---------------------------------------

Number of support vectors: 19

Ein: 0.002562

Eout: 0.023585

C=10^6 ---------------------------------------

Number of support vectors: 17

Ein: 0.000641

Eout: 0.023585

Therefore, the C=100 results in the lowest E_{out}.

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW8 Problem 9 & Problem 10

import numpy as np
from sklearn.svm import SVC   #Used for implementing SVM.

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array

def one_vs_one_label(data_array, digit1, digit2): #For Problem 5 and 6.
    labelled_data_array = []
    for i in range (0, len(data_array)):
        if data_array[i,0] == digit1:
            labelled_data_array.append(1)
        elif data_array[i,0] == digit2:
            labelled_data_array.append(-1)
    return labelled_data_array

def filter_rest_of_digits(data_array, digit1, digit2): #Used for Problem 5 and 6. (one versus one)
    filtered_data_array = []
    for i in range(0, len(data_array)):
        if data_array[i,0] == digit1:
            filtered_data_array.append(data_array[i])
        elif data_array[i,0] == digit2:
            filtered_data_array.append(data_array[i])
    return np.array(filtered_data_array)

def calculate_binary_error(g_x, f_x):
    error_count = 0
    for i in range (0,len(g_x)):
        if(g_x[i] != f_x[i]):
            error_count = error_count + 1
    return error_count/len(g_x)

train_data_array = extract_data("features.train.txt")  #extract data
test_data_array = extract_data("features.test.txt")
train_data_array_np = np.array(train_data_array)
test_data_array_np = np.array(test_data_array)
label_1_vs_5_train = one_vs_one_label(train_data_array_np,1,5) #label +1 or -1
label_1_vs_5_test = one_vs_one_label(test_data_array_np,1,5)

filtered_data_1_vs_5_train = filter_rest_of_digits(train_data_array_np, 1, 5)
filtered_data_1_vs_5_test = filter_rest_of_digits(test_data_array_np, 1, 5)

print("C=0.01 ------------------------------------")
clf_digit_1_and_5 = SVC(C=0.01, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=1 -------------------------------------")
clf_digit_1_and_5 = SVC(C=1, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=100 --------------------------------------")
clf_digit_1_and_5 = SVC(C=100, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=10^4 ----------------------------------------")
clf_digit_1_and_5 = SVC(C=10000, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("C=10^6 ----------------------------------------")
clf_digit_1_and_5 = SVC(C=1000000, kernel='rbf', degree=2, coef0=1.0,gamma=1.0) #kernel definition
clf_digit_1_and_5.fit(filtered_data_1_vs_5_train[:,1:],label_1_vs_5_train) #fit the SVM
label_1_vs_5_train_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_train[:,1:])
label_1_vs_5_test_predict = clf_digit_1_and_5.predict(filtered_data_1_vs_5_test[:,1:])
print("Number of support vectors:", len(clf_digit_1_and_5.support_)) #Number of support vectors
print("Ein: %2f" %calculate_binary_error(label_1_vs_5_train_predict, label_1_vs_5_train)) #calculate Ein
print("Eout: %2f" %calculate_binary_error(label_1_vs_5_test_predict, label_1_vs_5_test))  #calculate Eout

print("-----------------------------------------------")
```